Departamento de Ingeniería de Computadores
Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Gestor de Secretos Open-Source para Equipos de Trabajo

Estudiante: Moisés Romero García
Directores: Carlos Abalde Ramiro - Allenta Consulting S.L.
Guillermo López Taboada - Dpto. de Ingeniería de Computadores

A Coruña, 4 de septiembre de 2019.

A mi Padre, Madre, Hermana, Pareja, Amigos y Directores de proyecto

Agradecimientos

Agradecer a todas aquellas personas que me han estado apoyando a lo largo de esta etapa académica. En especial a mi padre Alejandro Romero por haberme permitido acercarme desde muy joven a todo lo relacionado con el mundo informático, a mi madre Dolores García por todo lo que ha hecho para permitirme estudiar y a aprovechar al máximo esta etapa y a mi hermana Alexandra Romero que aunque ya no está habitualmente en España me ha apoyado mucho. Mención especial a Nanuk por todo lo que me ha aportado simplemente con su presencia. Sin ellos no podría haber llegado hasta este punto de mi vida.

Gracias también a todos esos compañeros con los que he compartido estos cuatro años de mi vida y con los que he superado infinidad de prácticas y vivido muchos buenos momentos puesto que sin ellos no hubiera sido lo mismo. También aprovechar esta oportunidad para agradecer a mi pareja todo el apoyo que me ha dado a lo largo del grado y en especial a la paciencia que ha tenido en todo el tiempo que he dedicado al proyecto.

Agradecimientos al equipo de Allenta por permitirme realizar este proyecto en sus instalaciones.

Por último, agradecer a mis directores de proyecto Carlos Abalde Ramiro y Guillermo López Taboada, el haberme ayudado y guiado en el desarrollo de este proyecto que para mi significa un gran paso debido a la cantidad de tecnologías y conceptos nuevos que me ha permitido aprender.

Resumen

El objetivo principal de este Trabajo Fin de Grado ha sido la creación de un gestor de secretos *open-source* desde cero, tratando de solventar las carencias de los gestores más relevantes, añadiendo nuevas funcionalidades interesantes descubiertas tras el análisis de los mismos (5 *open-source* y 3 comerciales). Fruto de este análisis inicial, he decidido llevar a cabo el desarrollo de un gestor de secretos polivalente, es decir, que pueda ser utilizado por diversidad de departamentos, que permita almacenar más que contraseñas y, lo más importante en este proyecto; orientando su funcionalidad al uso que hacen los equipos de trabajo del mismo, con los requisitos de flexibilidad y granularidad que algo así exige en cuanto a los privilegios de acceso de cada usuario a los secretos almacenados.

La aplicación se ha desarrollado utilizando Python junto con el *framework* Django y el sistema de cifrado empleado ha sido PGP, en concreto se ha utilizado la implementación que ofrece GPG, definido bajo el estándar RFC-4880.

El diseño de la herramienta se ha basado en un enfoque centralizado con una arquitectura cliente/servidor clásica compuesta por dos aplicaciones. La funcionalidad del servidor queda expuesta mediante un API REST con la que se integran las múltiples instancias de la aplicación cliente. En lo que se refiere a la aplicación del lado servidor, el diseño se corresponde con el de una aplicación web clásica en la que cabe destacar nuevas características como la autenticación de clientes usando criptografía asimétrica, inclusión de una capa de caché basada en la base de datos clave-valor de alto rendimiento Redis, integración con el servicio de tareas asíncronas Celery, integración con APIs de terceros, uso de Sentry como sistema de detección de errores, etc.

Como resultado final se ha obtenido la aplicación **MrSecrets**, un nuevo gestor de secretos que mejora los proyectos previamente analizados y, que gracias a las necesidades de la empresa de mi director Carlos, se ha utilizado para reemplazar la herramienta actual permitiendo su aplicación en un entorno real. Además, ha sido validado en colaboración con la misma empresa y será publicado en GitHub bajo una licencia libre, en concreto la licencia GPL, para que cualquier entidad que necesite un gestor de secretos polivalente que base su cifrado en GPG y se especialice en ofrecer un gran nivel de granularidad a la hora de asignar permisos así como en la gestión de grupos, tenga una opción más a su disposición.

Palabras clave: Gestor de contraseñas, Contraseña, GPG, Python, Django, Arquitectura cliente/servidor.

Abstract

The principal objective of this Final Degree Project has been the creation of an open-source secret's manager from the beginning. With a secret manager's implementation created from scratch, it has been looking to meet the needs no present in the current managers, therefore, after analysing the data of the current managers (5 open-source and 3 commercial), some interesting additional functionalities have been added.

As a benefit of this initial analysis, I decided to perform a polyvalent secret's manager, in other words, a manager able to be used by many different departments, with the allowance to store more than passwords and, the more significant in this project; orientating its functionality to the use that the work's team do in it, with the flexibly requirements and granularity that something like this demand in terms of access privileges of each user to the stored secrets.

The application has been developed using Python along with Django's framework, and the encryption system used it has been PGP; more precisely, the implementation offered by GPG has been used, defined by the standard RFC-4880.

The tool design has been based in a centralised approach with a classic client/server architecture that consists of two applications. The functionality of the server is exposed by an API REST with the integration of multiple instances of the client application. In terms of the server side application, the design corresponds of a classic web application where it should be noted some added distinguishing elements that provide high value and utility to the application such as the authentication of clients using asymmetric cryptography, inclusion of a caching layer based on the database Key-value Redis, integration with the asynchronous Celery task's service, integration with third party APIs, Sentry use as an error detection system, etc.

As a final result it has been obtained the application **MrSecrets**, a new secret's manager who improves the previously analysed projects and, in base to the specific requirements of my director Carlos' company, it has been used to replace the current tool allowing its application in a real environment. In addition, it has been validated in collaboration with the same company and it will be published in GitHub under a free licence, specifically the GPL licence, making it available to any entity with the need of a polyvalent Secret's Manager who based their encryption in GPG and has been specialised offering a high level of granularity when assigning permissions as well as in the groups management, having one more extra option.

Keywords: Password manager, Password, GPG, Python, Django, Client/server architecture.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Hoy en día tras el gran crecimiento que ha sufrido internet y la gran cantidad de servicios que se pueden consumir en él, los usuarios están "obligados" a disponer de una cuenta por cada servicio al que quieran acceder, generando la necesidad de tener contraseñas robustas y diferentes para cada una de las cuentas. En el caso de las empresas esta necesidad es más compleja puesto que en el ámbito profesional no se trabaja solo con servicios *online* sino que también existe la necesidad de almacenar de forma segura claves SSH (*Secure SHell*), credenciales de acceso de repositorios de paquetes, licencias software, credenciales VPN (*Virtual Private Network*), etc. Por este motivo de aquí en adelante se utilizará el término de "secreto" para referirse a los recursos que se almacenan en un gestor y de igual forma, se hará referencia a los gestores como "gestores de secretos".

Para evitar tener que recordar todos y cada una de los secretos, además de asegurarse de que en el caso de las contraseñas son distintas entre si, se ha creado lo que se conoce como gestores de secretos. Estas herramientas permiten almacenar credenciales y demás información, evitando que tengamos que recordarlas. Hay que tener en cuenta que el uso de estos gestores introduce un SPOF (*Single Point Of Failure*) al centralizar en un único lugar el acceso a todos los secretos, pero más allá de alternativas parciales como SSOs (*Single Sign-On*) y tecnologías como OAuth, son la mejor práctica a día de hoy. Además, ofrecen funcionalidades adicionales como auto-completado, generación de contraseñas aleatorias, generación de OTPs (*One-Time Passwords*), etc. Con el uso de estas herramientas, en caso de utilizar criptografía simétrica (muy habitual en los gestores destinados a usuarios individuales), el número de contraseñas que debe recordar el usuario se reduce a una única contraseña denominada contraseña maestra. La contraseña maestra es la clave que permite el acceso a la base de datos del gestor. Algunas soluciones utilizan otros métodos basados en cifrado asimétrico como por ejemplo GPG (*GNU Privacy Guard*) o mecanismos biométricos complementarios para mitigar la debilidad que introduce la centralización.

Desde el punto de vista profesional, los gestores de secretos son una herramienta funda-

mental puesto que el número de cuentas utilizadas es elevado y suele ser habitual tener que actualizarlas cada poco tiempo además de que determinadas contraseñas deben poder ser accesibles por varios usuarios o grupos de usuarios. En la sección 2 se analizan las opciones más populares, categorizadas en dos grandes grupos, los gestores *open-source*, más centrados en el usuario individual, aunque también existen soluciones que tratan de dar soporte a equipos, y los comerciales, aunque también destinados a usuarios individuales, más orientados al uso profesional.

Gestores *open-source* como Gopass [1] y KeePass [2] suelen carecer de características que ofrecen los comerciales, además de que normalmente precisan de poca infraestructura (por ejemplo, un simple fichero .kdbx en el caso de KeePass) y su mantenimiento es llevado a cabo por quien lo utiliza. Por otro lado tenemos los comerciales, que ofrecen una gestión más personalizada y añaden funcionalidades como la gestión de contraseñas para grupos, posibilidad de acceso en caso de olvidar la contraseña maestra, etc, algunos ejemplos son 1Password [3], LastPass [4] o Dashlane [5]. Todos los gestores mencionados aquí, han sido analizados en detalle en las secciones 2.1 y 2.2.

Los gestores *open-source* ofrecen una serie de características que no son suficientes para todas las empresas, siendo su mayor carencia la gestión de grupos o equipos de trabajo, característica en la que se centran más los comerciales puesto que cubren casos de uso más profesionales. Sin embargo, como contrapartida, habitualmente su implementación no es auditable, cuestión especialmente delicada dada la criticidad de la información que están destinados a gestionar.

1.1 Motivación

Las principales motivaciones que han llevado a realizar este proyecto han sido el interés por el funcionamiento de los gestores de secretos y la necesidad que existía en Allenta Consulting S.L. [6] de evolucionar la gestión de secretos que utilizaban por otro que se adaptara mejor a sus nuevas necesidades, fruto de su crecimiento tanto en número de clientes como en número de empleados y la cantidad de roles de los mismos. Antes de nada se han realizado búsquedas de posibles soluciones ya existentes, pero finalmente se ha llegado a la conclusión de que no hay una solución que se adapte a sus necesidades específicas enumeradas en la sección de análisis de requisitos 3.1.

Puesto que el entorno de Allenta reúne las características del entorno al que va destinado el nuevo gestor de secretos, he decidido diseñar y desarrollar el nuevo gestor pensando en cubrir las necesidades de la empresa tratando de mejorar la herramienta empleada actualmente. Puesto que el entorno de referencia de uso para la herramienta ha sido la empresa Allenta, en el resto del documento se ha hecho referencia a esta en diferentes puntos.

El gestor se ha desarrollado pensando en los usuarios de entornos Linux y macOS que deberán gestionar no solo contraseñas, sino también otros elementos cuya gestión habitual sea especialmente costosa como por ejemplo: archivos de configuración, certificados, etc. Además, en vista de que no existe ninguna solución *open-source* equivalente a la descrita en este documento, el nuevo gestor será publicado bajo la licencia libre GPL, de forma que pueda ser utilizado y extendido por todo aquel que así lo desee.

1.2 Objetivos

El principal objetivo de este proyecto ha sido el diseño y la implementación de un gestor de secretos que mejore las soluciones existentes en la actualidad y que se centre en las necesidades particulares de Allenta, sustituyendo la herramienta usada durante los últimos años. Para esto se ha desarrollado la solución utilizando el lenguaje de programación Python [7] en su versión 3.6 con la ayuda del *framework* de desarrollo Django [8]. La arquitectura que ha seguido la nueva solución consta de dos partes, una aplicación a modo de servidor que expone un API REST (*Representational State Transfer*) y otra a modo de cliente en forma de CLI (Command Line Interface) que consulta el API (*Application Programming Interface*) del servidor permitiendo interactuar con el gestor. Al igual que en la herramienta actual, todo el cifrado está basado en el sistema que ofrece GPG definido bajo el estándar RFC4880 [9].

A continuación, se incluye el listado de objetivos concretos que se habían definido previamente al desarrollo. Estos objetivos han sido clasificados en función de su carácter en principales y secundarios:

- Objetivos principales:
  - Servidor:
    - Acceso remoto seguro, incluso en ausencia de una capa TLS (*Transport Layer Security*).
    - Definición de un modelo de secreto flexible, es decir, que los secretos puedan ser tanto simples claves como documentos de carácter general.
    - Organización jerárquica de los secretos.
    - Gestión flexible de usuarios y equipos de trabajo.
    - Soporte de un modelo de permisos propio, que permita restringir el acceso a secretos de forma sencilla.
    - Soporte de plantillas, a modo de esquema o modelo de datos de los secretos almacenados en el gestor.
  - Cliente:

    - Gestión remota de recursos (i.e. carpetas y secretos).
    - Acceso remoto a los secretos.
    - Modificación remota de los secretos.

- Objetivos secundarios:
    - Integración con sistemas de seguimiento de errores tipo Sentry.
    - Infraestructura para la ejecución de tareas asíncronas del lado servidor (p.e. borrados en masa, auditorías, etc.).
    - Soporte nativo para OTPs.
    - Auditoría periódica de los secretos almacenados en el lado servidor: identificación de contraseñas débiles, integración con sistemas de alerta de brechas de seguridad, etc.

Los objetivos principales conforman la estructura general y el núcleo de la solución, mientras que los objetivos secundarios añaden funcionalidad extra sin la cual el gestor sería plenamente funcional.

1.3 Metodología

La metodología empleada en el desarrollo de esta solución ha seguido las directrices dadas por el método de ingeniería, partiendo del estudio de herramientas similares tanto *open-source* como comerciales, recopilación de requisitos y limitaciones de la herramienta actual en base a entrevistas con el equipo de Allenta y continuando con las fases habituales de análisis, diseño e implementación. Las nuevas funcionalidades han sido añadidas de forma incremental tras validar las funcionalidades anteriores.

En primer lugar se ha realizado un estudio de la situación actual de los gestores de secretos analizando tanto gestores *open-source* como comerciales para tener una visión global de las características más comunes e interesantes para el proyecto.

Una vez conocida la situación y las soluciones existentes, se han recopilado requisitos y analizado las limitaciones de la herramienta actual para así saber qué posibles mejoras introducir en el nuevo gestor y tratar de que sea lo más amigable y similar al actual por comodidad para los usuarios.

Con una visión global de la situación y con una primera lista de requisitos, se han realizado pruebas con diferentes librerías Python que se han considerado interesantes. Para esto se ha preparado y creado un primer entorno de desarrollo de prueba.

Tras haberse familiarizado con las librerías existentes, se ha creado el entorno de desarrollo definitivo y dado comienzo a la etapa de diseño de la solución software. Posteriormente, se

ha procedido a la implementación del servidor y del cliente así como las pruebas pertinentes, con ciclos periódicos de validación de los prototipos con el equipo de Allenta.

Finalmente, se han ido añadiendo distintas funcionalidades que aumentan el alcance de la herramienta. Con la solución completa, se han realizado las pruebas automatizadas y por último se ha llevado a cabo un despliegue del software en AWS (Amazon Web Services), obteniendo una instalación real preliminar del software totalmente funcional.

1.4 Planificación

La planificación del proyecto se ha realizado estableciendo como fecha de inicio el 04/02/2019 y fecha de fin estimada el 18/09/2019. En la figura 1.1, se muestra el diagrama de Gantt correspondiente a la planificación inicial.

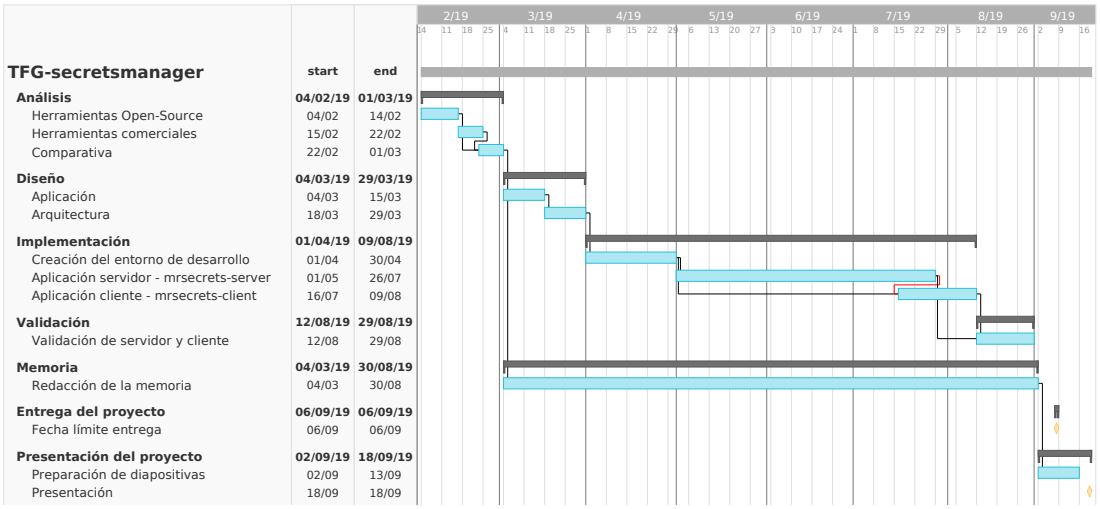

Figura 1.1: Diagrama de Gantt de la planificación inicial

Al terminar el proyecto el estado del diagrama de Gantt ha sido el de la figura 1.2. Como se pueden observar, ha habido una serie de desviaciones leves en la fase de implementación de la aplicación. Estas desviaciones han sido debidas a la época de exámenes y a la reducción del tiempo disponible durante los meses de junio y julio por motivos de trabajo. Para reducir lo máximo posible el impacto de estas pausas se han comenzado antes las fases de implementación del servidor, implementación del cliente y gracias a la sobre-estimación de tiempo en la tarea de validación, solo se ha sobrepasado la planificación por motivos referentes a la elaboración de la memoria.

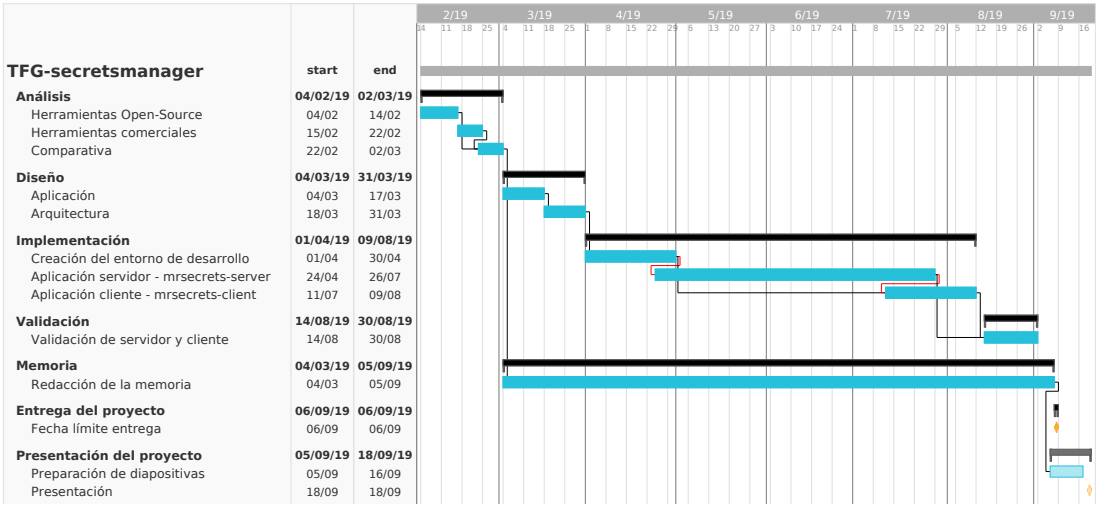

Figura 1.2: Diagrama de Gantt de la planificación finalizar el proyecto

1.5 Costes del proyecto

En lo referente a los costes del proyecto, se han realizado estimaciones teniendo en cuenta la participación de tres empleados, el autor del documento y los dos directores. El autor ha asumido los roles de analista, administrador de sistemas y desarrollador. Por su parte los directores del proyecto han tomado rol de directores de proyecto.

El cuadro 1.1, refleja las estimaciones de costes de la realización del proyecto teniendo en cuenta los tres empleados citados, con una media de 4 horas diarias de trabajo y sin contar fines de semana.

Rol	€/hora	Horas día	Horas Total	Coste Total (€)
Director de proyecto 1	45	-	50	2.250
Director de proyecto 2	45	-	15	675
Analista	15	4	84	1.260
Admin. de sistemas	15	4	160	2.400
Desarrollador	15	4	404	6.060
Total			**673**	**12.645**

Cuadro 1.1: Desglose de costes de los empleados del proyecto

Por otro lado, los costes materiales que ha supuesto han sido: ordenador portátil con el que se ha llevado a cabo el proyecto, con un coste de **1.150€**, y el uso de servicios de infraestructura en la nube de Amazon, para el despliegue de la aplicación, con un coste de **56.43€** utilizando una instancia t3.large de AWS con Linux.

Por lo tanto el coste total del proyecto a ha sido de **14.851,43€**.

1.6 Estructura de la memoria

La estructura de la memoria de este proyecto está dividida en capítulos tal y como se muestra a continuación:

- **Capítulo 1:** introduce y da una visión global del proyecto mediante una introducción, la motivación que llevó a la realización de este proyecto, los objetivos a cumplir, la metodología empleada, la planificación seguida durante la realización del proyecto, los costes que ha ocasionado el mismo y la descripción de cómo está estructurada la memoria.
- **Capítulo 2:** se expone el estado actual de los gestores de secretos, citando algunos de los más relevantes en la actualidad. Posteriormente se hace una crítica constructiva que justifica la realización de este TFG, finalizando con la explicación de la solución propuesta.
- **Capítulo 3:** se analiza el problema a resolver, mediante el análisis de requisitos, el análisis de la solución propuesta y el análisis de la seguridad del gestor de secretos.
- **Capítulo 4:** se explican las decisiones de diseño y se comentan las herramientas y librerías utilizadas así como la arquitectura software empleada.
- **Capítulo 5:** se comentan detalles de implementación que resultan críticos o cuya complejidad es tal que merecen ser explicados.
- **Capítulo 6:** conclusiones obtenidas y lecciones aprendidas durante la realización del proyecto.
- **Capítulo 7:** listado de ampliaciones o mejoras de la solución software junto con recomendaciones de cómo realizarlas.

Capítulo 2

Estado del arte

El estado en el que se encuentran los gestores de contraseñas es muy variado dependiendo de qué herramienta hablemos, siendo algunos de los puntos más diferenciales: el tipo de base de datos que utilizan para almacenar la información [10], los algoritmos criptográficos empleados para ocultar y proteger la información, o la ubicación en la que se almacenan las bases de datos, documentos y demás sistemas utilizados para guardar los secretos. Para tener una mejor idea de la situación actual, se ha realizado un análisis detallado de algunos de los gestores más utilizados en la actualidad; han sido divididos en dos grupos en función de la naturaleza de su código, diferenciando así entre gestores *open-source* y gestores comerciales, situando en *open-source* todos aquellos que exponen su código al público y en comerciales los que mantienen su código de forma privada y solamente ofrecen el uso del gestor bien sea de forma gratuita o de pago.

Una de las mayores ventajas de los gestores *open-source* es la posibilidad de realizar una auditoría completa sobre los mismos, permitiendo así su análisis en profundidad y, por ejemplo, la elaboración de pruebas para certificar puntos interesantes como su nivel de seguridad, correcto uso del sistema de cifrado en el que se basan y demás puntos que se consideren relevantes.

A continuación se expondrán los aspectos positivos y negativos que se han encontrado en las herramientas que se han analizado.

2.1 Gestores *open-source*

2.1.1 Password-Store[11]

Esta solución es la utilizada por Allenta. Se trata de un gestor muy simple que sigue la filosofía Unix tratando de simplificar la gestión de los secretos y que basa su seguridad en el uso de criptografía asimétrica, en concreto la implementación de GPG. Su estructura consiste en un sistema de ficheros compuesto por documentos cifrados por una o más claves públi-

cas, siendo cada documento un secreto. Esta estructuración permite que el gestor pueda ser utilizado por un mismo grupo de usuarios mediante la utilización de Git o NFS (*Network File System*) entre otras tecnologías. Además, en el caso de utilizar Git, se obtienen las ventajas propias de una herramienta de versionando, como persistencia de los datos y la existencia de un histórico basado en los *commits* que se realizan.

- Aspectos positivos:
  - Basa su sistema de cifrado en la criptografía asimétrica bajo la implementación de GPG.
  - Requiere poca infraestructura para funcionar pues su estructura se compone de carpetas y documentos cifrados.
  - Ofrece muchas funcionalidades extra mediante la instalación de extensiones (p.e. soporte para OTPs).
- Aspectos negativos:
  - El concepto de secreto para este gestor no incluye los documentos de propósito general (i.e. no permite almacenar cualquier tipo de documento).
  - Tras actualizar los permisos (i.e. definir quién puede descifrar un documento con su clave privada) sobre un conjunto de documentos determinado, es necesario volver a cifrar todos los documentos en función de las claves públicas que tengan asignadas.
  - No ofrece soporte directo para la creación y gestión de equipos. En Allenta, para poder gestionar equipos, se utilizan documentos ".acl" a modo de *hack* que permiten definir grupos de usuarios y el acceso de los mismos a los documentos.
  - La estructura completa de secretos puede ser vista por todos los usuarios independientemente de que no puedan descifrarlos. Es decir, por su naturaleza basada en un sistema de archivos, no se puede hacer que un usuario solo visualice aquellos documentos a los que tiene acceso.

2.1.2 Gopass[1]

Se trata de un gestor cuyo objetivo es mejorar la propuesta de Password-Store (sección 2.1.1, página 9) manteniendo el sistema de cifrado del mismo así como el CLI. Su código ha sido escrito en el lenguaje de programación Go, mantiene la misma forma de estructurar los secretos que su predecesor y se centra en mejorar, sobre todo, lo referente a la gestión de equipos y grupos de trabajo. Los secretos son agrupados bajo lo que en Gopass se denomina *store*.

- Aspectos positivos:
  - Permite gestionar los usuarios que utilizan un *store* concreto.
  - En el momento de la creación de un secreto se puede elegir qué usuarios tienen acceso al mismo (i.e. con qué claves públicas se cifrará el secreto).
  - Ofrece soporte para equipos, permitiendo indicar los usuarios que pueden acceder a cada secreto así como editarlos.
  - Ofrece soporte para OTPs de forma nativa.
  - Para este gestor el término secreto incluye tanto las claves tradicionales (i.e. conjunto de campos) como binarios permitiendo almacenar documentos de carácter general (p.e. PDFs, imágenes, etc).
- Aspectos negativos:
  - A pesar de incorporar soporte para equipos, este sigue siendo insuficiente pues se necesita un *store* por cada equipo, entendiendo como equipo al conjunto de usuarios que pueden acceder a un *store*. No es una aproximación realista para los casos de uso más comunes.
  - La mayoría de sus funcionalidades se encuentran en estado *alpha* y *beta*.
  - Pierde la posibilidad de utilizar extensiones que ofrece su predecesor.

2.1.3 KeePass[2]

Es uno de los gestores de secretos *open-source* más utilizados a nivel personal y no está desarrollado pensando en su uso desde el punto de vista de equipos de trabajo. Utiliza criptografía simétrica, en concreto los algoritmos AES y TwoFish, para el cifrado de los datos que son almacenados en una base de datos especial con formato kdb o kdbx [12]. Gracias al uso de este tipo de sistema criptográfico, solo es necesario recordar la contraseña maestra (contraseña con la que se cifra la base de datos), utilizar un documento clave (documento que se utiliza para cifrar la base de datos) o ambas para tener acceso instantáneo a todos los secretos almacenados en la base de datos.

- Aspectos positivos:
  - Ofrece un control total sobre el documento de base de datos que almacena las claves pues es el usuario el encargado de ubicarlo y sincronizarlo entre dispositivos de la forma que considere oportuna (p.e. uso de un *pendrive*, Git, servicios externos como Dropbox o Google Drive, etc).
- Aspectos negativos:

  - Su enfoque es eminentemente doméstico y, por lo tanto, no ha sido desarrollado para ser utilizado por equipos de trabajo o grupos de usuarios.

2.1.4 Bitwarden[13]

Este gestor tienes dos versiones, la versión gratuita pensada para usuarios individuales y varias versiones de pago entre las que se encuentra una destinada al sector empresarial donde se incluyen funcionalidades como gestión de equipos, sincronización de directorios, etc. Sigue la arquitectura cliente-servidor y ofrece distintas formas para gestionar la infraestructura necesaria. Como algoritmos de cifrado para las conexiones entre el cliente y el servidor utiliza el algoritmo AES-256, hace *hashing* mediante el uso de un *salt* y PBKDF2 SHA-256.

Bitwarden ha expuesto los resultados de las auditorías realizadas a su producto [14] y además tiene un programa de recompensa de errores en HackerOne [15] para incentivar a los expertos a tratar de encontrar vulnerabilidades para así poder corregirlas.

- Aspectos positivos:
  - Ofrece las ventajas típicas de utilizar una estructura cliente-servidor como acceso centralizado, el cliente solo puede ver y acceder a los recursos del servidor a los que tiene acceso, etc.
  - Ofrecen la posibilidad de no utilizar su nube y en su lugar desplegar la infraestructura necesaria para alojar el servidor, mediante el uso de Docker, pasando a estar todo gestionado por el usuario.
  - Además de contraseñas permite guardar documentos de carácter general añadiéndolos como adjuntos a un secreto [16].
- Aspectos negativos:
  - Si se decide utilizar su nube, los datos se almacenan en los servidores de Microsoft Azure perdiendo el control de los mismos pero delegando en expertos en seguridad el tratamiento de estos.

2.1.5 LessPass[17]

Este gestor se diferencia de los demás en que utiliza la contraseña maestra e información conocida para generar la contraseña de tal modo que no requiere almacenar las contraseñas. La idea que propone es muy interesante y alejada de la forma habitual de generar una contraseña y aporta varias ventajas.

Además, se trata de un gestor *open-source* que al generar las contraseñas siempre de la misma manera no requiere sincronización de las contraseñas en los distintos dispositivos en los que se quiera utilizar.

- Aspectos positivos:
  - No requiere almacenar las contraseñas en una base de datos.
  - No se tiene que sincronizar con los dispositivos en los que se quiera utilizar.
- Aspectos negativos:
  - No permite almacenar documentos de propósito general.
  - No ofrece soporte para la creación de grupos de trabajo.

2.2 Gestores comerciales

De forma general hemos de destacar que el mero hecho de confiar nuestra información secreta (i.e. claves, cuentas, documentos confidenciales, etc) a una entidad externa, implica ceder a esa entidad la responsabilidad de administrar y proteger aquellos datos que queremos mantener seguros. En este aspecto, las soluciones *open-source* resultan más atractivas pues no ocultan detalles referentes al tratamiento de los datos como por ejemplo: los métodos que utilizan para almacenarlos, la forma de gestionarlos, cifrarlos y demás factores importantes que al utilizar un gestor comercial son confiados a la entidad que se encarga de gestionar su propio producto.

Los tres gestores que han sido analizados a continuación utilizan criptografía simétrica (en concreto el algoritmo AES-256) mediante el uso de una contraseña maestra para permitir el acceso a los datos que almacenan.

2.2.1 1Password[3]

Es uno de los gestores más conocidos y utilizados entre todas las soluciones de pago que existen. Ofrece soluciones tanto para el uso individual como empresarial, centrándose en este último, siendo GitLab [18] un ejemplo representativo de este último grupo.

En 1Password utilizan el término *vault* para definir un conjunto de secretos que pertenecen a un mismo grupo. Los *vaults* pueden ser almacenados o no en la nube mediante el uso de los servicios de sincronización que admite 1Password [19], siendo estos: la propia cuenta de 1Password, Dropbox o iCloud. Como alternativa al uso de estos servicios es posible utilizar un servidor WLAN (*Wireless Local Area Network*) propio para la sincronización.

- Aspectos positivos:
  - Soporte para OTPs nativo.
  - Ofrece soporte para la gestión de grupos basada en *vaults*.

  - Por defecto, la base de datos que contiene los secretos no es almacenada en la nube.
  - Permite hacer búsqueda de secretos en múltiples *vaults* de forma simultánea.
  - Realiza comprobaciones sobre las claves almacenadas alertando al usuario si se considera que son débiles, que han sido comprometidas, etc.
  - Además de contraseñas permite guardar documentos de carácter general añadiéndolos como adjuntos a un secreto.

- Aspectos negativos:
  - No existe una aplicación nativa para Linux.
  - Como la asignación de permisos para acceder a los recursos se basa en el uso de *vaults*, no ofrece suficiente granularidad para la asignación de los mismos.

2.2.2 LastPass[4]

LastPass es otro gestor muy conocido que ofrece varias versiones entre la que se encuentra la dedicada al entorno empresarial y en la que se ha centrado el siguiente listado de aspectos positivos y negativos.

- Aspectos positivos:
  - Ofrece un buen sistema para la gestión de equipos de trabajo permitiendo asignar grupos de contraseñas en función de equipos o incluso de proyectos.
  - Permite un control muy detallado sobre la asignación de usuarios a los secretos almacenados.

- Aspectos negativos:
  - Ya han sido detectadas múltiples vulnerabilidades [20] que han puesto en riesgo las claves de los usuarios.

2.2.3 Dashlane[5]

Al igual que los dos anteriores ofrece una versión pensada para el entorno empresarial cuyos aspectos positivos y negativos han sido expuestos a continuación.

- Aspectos positivos:
  - Ofrece un buen sistema para la gestión de equipos de trabajo permitiendo crear grupos de usuarios para posteriormente asignar y gestionar su acceso a los recursos existentes.

  - Ofrece la posibilidad de habilitar alertas de seguridad que notificarán al usuario en caso de que alguna de las páginas en las cuales tenga cuenta haya sido atacada o haya sufrido alguna brecha de seguridad.
  - Además de contraseñas permite guardar documentos de carácter general.
  - Ofrece la posibilidad de almacenar los datos en local en lugar de en sus servidores.
  - Envía alertas instantáneas si una de las cuentas almacenadas está en riesgo o comprometida (teniendo en cuenta el dominio al que pertenece la cuenta).

- Aspectos negativos:

  - No ofrece una aplicación para su uso en Linux. Solo se puede utilizar mediante extensiones en el navegador, con los riesgos que eso implica [21].

2.3 Crítica del estado del arte

Habiendo analizado los gestores más populares, se ha podido determinar una serie de puntos que son claves a la hora de escoger uno u otro y que han servido como inspiración para la creación del nuevo gestor. Uno de los puntos y quizás el más importante es la poca granularidad a la hora de gestionar los permisos y los grupos de usuarios. Centrándonos más en la seguridad, cabe destacar también la imposibilidad de realizar auditorías de seguridad sobre las herramientas no *open-source* a las que se confían los secretos. Otro inconveniente muy importante es la ubicación en la que se almacenan los secretos; lo ideal es tener control total sobre esta ubicación, pero la mayoría de los gestores no ofrecen esta posibilidad. Otro aspecto poco común entre los gestores analizados es la posibilidad de almacenar todo tipo de recursos como documentos de carácter general. Por último, aunque las alternativas que existen son muy variadas, hay muy pocas que basen su sistema de cifrado en el uso de criptografía asimétrica.

Como se ha mencionado en el análisis de Password-Store (sección 2.1.1, página 9), el soporte que ofrece para la gestión de equipos es muy pobre y aunque la adaptación realizada por Allenta permite una mejor gestión de los mismos, no es suficiente. Por otra parte el gestor Gopass aunque soluciona parte del problema, no permite una gestión tan personalizada como la que se busca ya que su solución a la gestión de equipos implica tener un *store* de secretos para cada equipo concreto, junto con un repositorio Git para cada *store*. Además, hay que tener en cuenta que gran parte de las funcionalidades que ofrece están en *beta*. Por otro lado la propuesta de LessPass (sección 2.1.5, página 12) resulta ciertamente interesante pero no se adapta al entorno objetivo.

De todos los gestores que han sido analizados el que ofrece una aproximación más cercana a lo deseado por Allenta es Bitwarden (sección 2.1.4, página 12) mediante el despliegue de la

infraestructura necesaria a través de Docker. Sin embargo, Bitwarden, además de excesivamente complejo, no ofrece la posibilidad de utilizar GPG como sistema de cifrado, por lo que no cumple con uno de los requisitos fundamentales.

En el cuadro 2.1, se puede apreciar una matriz comparativa en la que se exponen una serie de características y se indica para cada gestor de los analizados si la cumple o no. Los nombres de los gestores se han abreviado y las correspondencias abreviatura - nombre son las siguientes: **PS**=Password-Store, **GP**=GoPass, **KP**=KeePass, **BW**=BitWarden, **LessP**=LessPass, **1P**=1Password, **LastP**=LastPass y **DL**=DashLane.

Características	PS	GP	KP	BW	LessP	1P	LastP	DL
Soporte GPG	✓	✓						
Soporte Linux	✓	✓	✓	✓				
Concepto de secreto		✓		✓		✓	✓	✓
Comprobación de contraseñas		✓		✓		✓	✓	✓
Control muy detallado de permisos				✓			✓	✓
Control total sobre el servidor	✓	✓	✓	✓		✓		✓
Soporte para grupos de usuarios		✓		✓		✓	✓	✓
Soporte nativo para OTPs		✓		✓		✓	✓	✓

Cuadro 2.1: Matriz de características requeridas y gestores de contraseñas

Como solución al problema planteado en Allenta y para ayudar a tomar un enfoque global de la situación se han analizado los gestores actuales en busca de uno que cumpliera con sus necesidades. Teniendo en cuenta los puntos ya mencionados, y tras haber identificado claramente la inexistencia de un gestor de secretos que cumpla con las características mencionadas en los objetivos de la sección 1.2, se ha creado un gestor de secretos adaptado a sus necesidades.

2.4 Propuesta

Debido a la situación actual de escasez de gestores *open-source* que ofrezcan flexibilidad suficiente a la hora de gestionar los permisos de cada usuario o grupo, permitan almacenar documentos de carácter general a la vez que claves, y basen su sistema de cifrado en el uso de criptografía asimétrica, se ha decidido realizar el desarrollo de un gestor de secretos desde cero. Como el entorno de Allenta se adapta perfectamente al perfil al que va destinado el gestor de secretos, se ha decido utilizar la empresa como entorno de referencia.

Puesto que el gestor se ha creado desde cero, además de haber utilizado la implementación de cifrado asimétrico de GPG, se ha aprovechado para personalizar el funcionamiento del gestor y adaptarlo a las necesidades específicas de Allenta. Puesto que se pretende que este proyecto sea *open-source*, podrá ser utilizado por quién necesite un gestor centrado en el ámbito empresarial con GPG como método de cifrado.

Este nuevo gestor que ha sido nombrado **MrSecrets**, agrupa algunas de las funcionalidades más interesantes de los gestores analizados. Es una nueva propuesta específicamente centrada en su aplicación en un ámbito empresarial donde existan varios equipos de usuarios diferenciados, y donde no todos tendrán permiso para acceder a todos los secretos. Las funcionalidades más interesantes que se han añadido y que ya ofrecen otros gestores son:

- Utilización del concepto de secreto en su completitud, englobando tanto claves como cualquier tipo de documento, al igual que los gestores analizados en las secciones 2.1.2 (Gopass), 2.1.4 (Bitwarden), 2.2.1 (1Password), 2.2.2 (LastPass) y 2.2.3 (Dashlane).
- Estructura basada en la arquitectura cliente-servidor, al igual que la solución de Bitwarden (sección 2.1.4, página 12).
- Soporte nativo para OTPs, como en el caso de 1Password (sección 2.2.1, página 13).
- Integración con sistemas de alertas mediante su integración con servicios proporcionados por terceros, al igual que 1Password (sección 2.2.1, página 13) y Dashlane (sección 2.2.3, página 14).
- Control muy detallado sobre los permisos, tanto de usuarios individuales como de grupos, característica común en los gestores LastPass (sección 2.2.2, página 14) y Dashlane (sección 2.2.3, página 14) en sus versiones empresariales.

Es importante destacar que la funcionalidad de auto-completado de datos de formularios de *login* no ha sido citada ni tomada en cuenta. Tanto por sus potenciales inconvenientes [22] como por ser un caso de uso irrelevante para Allenta.

En conclusión, la propuesta es desarrollar una solución que, tomando características de varias de las alternativas existentes analizadas en el capítulo 2, y teniendo presentes los objetivos enumerados en la sección 1.2, unifique todas las características y funcionalidades en un gestor de secretos centrado en dar soporte a las necesidades de entornos como el de Allenta y que a su vez sea sencillo, fiable, suficientemente versátil y abierto a la comunidad mediante una licencia libre.

Añadiendo este nuevo gestor de secretos denominado **MrSecrets (MrS)** al cuadro 2.1, se ha obtenido el cuadro 2.2, en el cual se puede apreciar que este cumple con todos las características especificadas.

Características	PS	GP	KP	BW	LessP	1P	LastP	DL	MrS
Soporte GPG	✓	✓							✓
Soporte Linux	✓	✓	✓	✓					✓
Concepto de secreto		✓		✓		✓	✓	✓	✓
Comprobación de contraseñas		✓		✓		✓	✓	✓	✓
Control muy detallado de permisos				✓			✓	✓	✓
Control total sobre el servidor	✓	✓	✓	✓		✓		✓	✓
Soporte para grupos de usuarios		✓		✓		✓	✓	✓	✓
Soporte nativo para OTPs		✓		✓		✓	✓	✓	✓

Cuadro 2.2: Matriz completa de características requeridas y gestores de contraseñas

Capítulo 3

Análisis del problema

En este capítulo se ha realizado un análisis de requisitos del gestor de secretos. Estos requisitos han sido categorizados en cuatro grupos: requisitos técnicos, requisitos de administración, requisitos de funcionalidades extra y requisitos de seguridad y de interfaz.

También se han identificado los casos de uso posibles y se han clasificado en casos de uso del administrador, casos de uso del usuario normal y casos de uso del gestor de secretos. Finalmente se han detallado las posibles soluciones al problema y posteriormente, teniendo en cuenta los requisitos citados, se ha aclarado cual ha sido la solución elegida y por qué.

Por último se ha realizado un breve análisis de seguridad respecto a la solución escogida en el que se detallan algunos inconvenientes de la estructura así como posibles soluciones a los mismos.

3.1 Análisis de requisitos

En esta sección se han especificado los requisitos que debe cumplir el gestor de secretos y que han sido recopilados mediante diversas reuniones con el equipo de Allenta.

La exposición y análisis de los requisitos se ha hecho mediante el uso de un cuadro por requisito que sigue el formato del cuadro 3.1.

Identificador	
Descripción	
Necesidad	
Prioridad	

Cuadro 3.1: Cuadro de requisitos

- **Identificador:** identificador único del requisito propuesto. Todos los descriptores han seguido el siguiente formato, R-[número del requisito], siendo el número de requisitos un valor de dos dígitos y comenzando por el 01.

- **Descripción:** descripción del requisito que esta siendo definido.
- **Necesidad:** nivel de importancia de la existencia de este requisito en el gestor. Se ha realizado una división en dos niveles:
  - **Obligatorio:** el requisito debe ser satisfecho por el gestor final.
  - **Deseable:** el requisito no es determinante para el gestor final.
- **Prioridad:** define el nivel de prioridad de diseño e implementación del requisito especificado. Los niveles de prioridad utilizados han sido: alta, media y baja.

3.1.1 Requisitos técnicos

Los cuadros 3.2 a 3.8 detallan los requisitos técnicos referentes al gestor.

R-01	
Descripción	El sistema de cifrado utilizado por el gestor será GPG.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.2: R-01, sistema de cifrado.

R-02	
Descripción	El gestor deberá permitir almacenar tanto claves como documentos de propósito general.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.3: R-02, almacenamiento de secretos.

R-03	
Descripción	Deberá existir un modelo de permisos flexible para definir el acceso de los usuarios y grupos a los secretos existentes.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.4: R-03, modelo de permisos flexible.

R-04	
Descripción	Al listar los recursos del gestor, se mostrará una columna en la que se indique la plantilla asociada a la carpeta correspondiente.
Necesidad	Obligatoria
Prioridad	Baja

Cuadro 3.5: R-04, mostrar plantilla al listar.

R-05	
Descripción	Al listar los recursos del gestor, se mostrará una columna en la que se indique si una contraseña es débil. Este campo solo aparecerá con valor para los secretos que contengan una contraseña.
Necesidad	Deseable
Prioridad	Baja

Cuadro 3.6: R-05, mostrar cuando una contraseña es débil.

R-06	
Descripción	Al listar los recursos del gestor, se mostrará una columna en la que se indique si el dominio al que pertenece un secreto ha sufrido alguna brecha de seguridad. Este campo solo aparecerá con valor para los secretos que contengan un dominio.
Necesidad	Deseable
Prioridad	Baja

Cuadro 3.7: R-06, mostrar cuando un dominio ha sido comprometido.

R-07	
Descripción	El gestor funcionará en sistemas basados en Linux.
Necesidad	Obligatoria
Prioridad	Alta

Cuadro 3.8: R-07, funcionamiento en Linux.

3.1.2 Requisitos de administración

Los cuadros 3.9 a 3.15 detallan los requisitos de administración referentes al gestor, en los que se especifican las funcionalidades que podrán ser llevadas a cabo por un administrador.

R-08	
Descripción	El gestor deberá permitir crear usuarios especificando nombre de *login*, nombre completo y archivo que contiene la clave pública del mismo.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.9: R-08, creación de usuarios.

R-09	
Descripción	El gestor debe permitir crear grupos especificando nombre del grupo, descripción y opcionalmente miembros.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.10: R-09, creación de grupos.

R-10	
Descripción	El gestor debe permitir crear plantillas especificando nombre de la plantilla y descripción.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.11: R-10, creación de plantillas.

R-11	
Descripción	Los usuarios, grupos y plantillas podrán ser eliminados.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.12: R-11, gestión de usuarios, grupos y plantillas.

R-12	
Descripción	Se deberá poder editar, enlazar y desenlazar las plantillas con carpetas.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.13: R-12, edición y acciones posibles sobre plantillas.

R-13	
Descripción	El gestor permitirá obtener un listado de usuarios, grupos y plantillas existentes.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.14: R-13, listado de usuarios, grupos y plantillas.

R-14	
Descripción	La gestión de los permisos (i.e. asignación y revocación) debe ser sencilla pero lo suficientemente compleja para gestionar permisos de usuarios y grupos en conjunto.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.15: R-14, gestión de permisos.

3.1.3 Requisitos de usuario

Los cuadros 3.16 a 3.22 detallan los requisitos de usuario referentes al gestor, en los que se especifican las funcionalidades que podrán ser llevadas a cabo por un usuario.

R-15	
Descripción	Los usuarios deben poder obtener un listado de todos los secretos a los que tienen acceso.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.16: R-15, listado de secretos en función de los permisos.

R-16	
Descripción	Deberá poderse listar los recursos de una carpeta en concreto.
Necesidad	Obligatorio
Prioridad	Media

Cuadro 3.17: R-16, listar recursos de una carpeta.

R-17	
Descripción	Deberá poderse listar las plantillas existentes en el gestor.
Necesidad	Obligatorio
Prioridad	Media

Cuadro 3.18: R-17, listar las plantillas existentes.

R-18	
Descripción	Los usuarios deben poder crear y eliminar carpetas y secretos.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.19: R-18, creación y eliminación de carpetas y secretos.

R-19	
Descripción	Los usuarios deben poder editar secretos en función de sus permisos.
Necesidad	Obligatorio
Prioridad	Alta

Cuadro 3.20: R-19, edición de secretos.

R-20	
Descripción	El usuario podrá recuperar un secreto cuyo contenido se mostrará en el terminal. Además también podrá elegir guardar el secreto recuperado en un documento.
Necesidad	Obligatorio
Prioridad	Media

Cuadro 3.21: R-20, obtención de un secreto.

R-21	
Descripción	El usuario podrá eliminar recursos (i.e. carpetas o secretos) de forma asíncrona.
Necesidad	Obligatorio
Prioridad	Media

Cuadro 3.22: R-21, eliminación asíncrona de recursos.

3.1.4 Requisitos de funcionalidades extra

Los cuadros 3.23 a 3.26 detallan los requisitos de funcionalidades extra referentes al gestor, en los que se especifican las funcionalidades extra que incorporará y sin las cuales sería totalmente funcional.

R-22	
Descripción	El gestor realizará comprobaciones para asegurarse de que los archivos existentes en el gestor existan en la BD (Base de Datos) y eliminará aquellos que ya no se encuentren registrados.
Necesidad	Deseable
Prioridad	Media

Cuadro 3.23: R-22, purgado de archivos basura.

R-23	
Descripción	El gestor realizará comprobaciones automáticas de la seguridad de las contraseñas almacenadas. Para realizar estas comprobaciones se permitirá establecer un conjunto de condiciones que definen una contraseña como fuerte.
Necesidad	Deseable
Prioridad	Media

Cuadro 3.24: R-23, revisión automática de la seguridad de las contraseñas.

R-24	
Descripción	El gestor realizará comprobaciones automáticas de los dominios que encuentre en los secretos almacenados, comprobando si han sufrido alguna brecha de seguridad recientemente o no.
Necesidad	Deseable
Prioridad	Baja

Cuadro 3.25: R-24, revisión automática de los dominios almacenados en los secretos.

R-25	
Descripción	El gestor ofrecerá soporte para la generación de OTPs. Tanto TOTPs (*Time-based One-time Password*) como HOTPs (*HMAC-based One-time Password*).
Necesidad	Deseable
Prioridad	Baja

Cuadro 3.26: R-25, generación de OTPs.

3.1.5 Requisitos de interfaz y de seguridad

Los cuadros 3.27 a 3.29 detallan los requisitos referentes a la interfaz del gestor y a la seguridad del mismo en temas de comunicación entre cliente y servidor.

R-26	
Descripción	La interfaz del gestor deberá ser un CLI.
Necesidad	Obligatoria
Prioridad	Alta

Cuadro 3.27: R-26, interfaz CLI.

R-27	
Descripción	Para poder acceder a los secretos el usuario debe estar registrado en el mismo y utilizar un *token* de usuario cifrado con la clave pública del servidor y firmado por su clave privada.
Necesidad	Obligatoria
Prioridad	Alta

Cuadro 3.28: R-27, autenticación mediante un *token* y uso de GPG.

R-28	
Descripción	Las comunicaciones entre el gestor y el usuario usarán GPG como método de cifrado, cifrando las peticiones del cliente al servidor con la clave pública del servidor y las respuestas del servidor con la clave pública del usuario. Además, se dará soporte para el uso de TLS.
Necesidad	Obligatoria
Prioridad	Alta

Cuadro 3.29: R-28, comunicaciones seguras entre cliente y servidor mediante GPG.

3.2 Casos de uso

Los casos de uso que se han encontrado en el gestor se muestran en el diagrama expuesto en la figura 3.1.

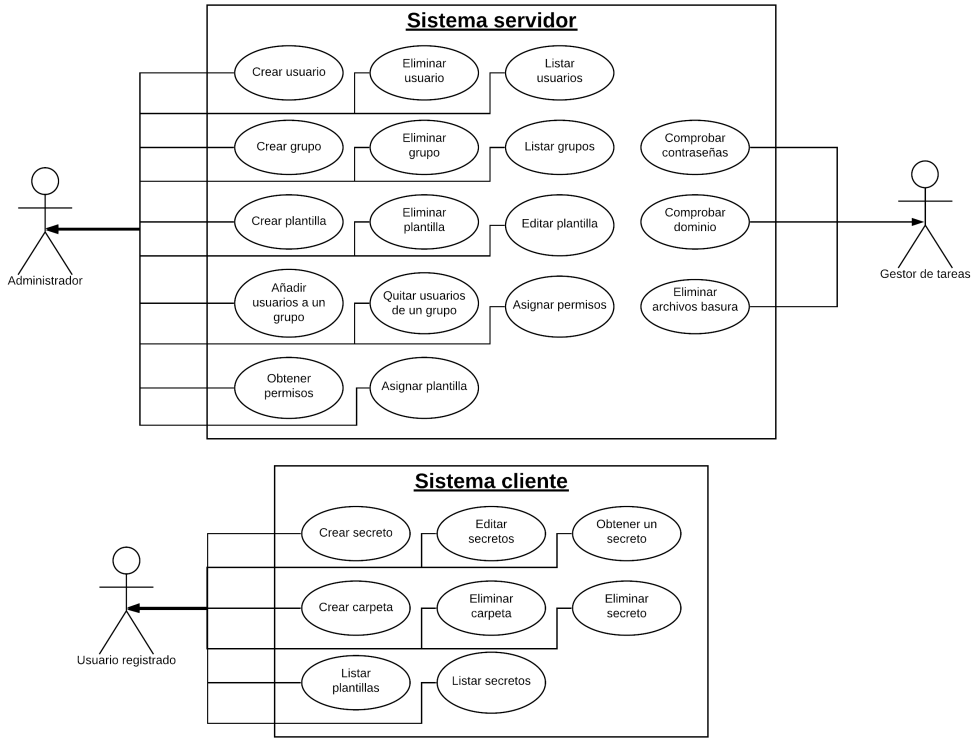

Figura 3.1: Diagrama de casos de uso

Del cuadro 3.31 al 3.55 se han detallado los casos de uso que aparecen en la figura 3.1. Por coherencia con la sección 3.1 se describirán los casos de uso en un cuadro. El formato que seguirán es el del cuadro 3.30.

Identificador	
Caso de uso	
Actores	
Precondiciones	
Secuencia normal	
Postcondiciones	
Excepciones	

Cuadro 3.30: Cuadro de casos de uso

- **Identificador:** identificador único del caso de uso a tratar. Todos los descriptores han

seguido el siguiente formato, CU-[número del caso de uso], siendo el número de caso de uso un valor de dos dígitos y comenzando por el 01.

- **Actores:** actores que intervienen en el caso de uso.
- **Precondiciones:** condiciones iniciales que debe cumplir el sistema para poder llevar a cabo el caso de uso.
- **Secuencia normal:** conjunto de pasos que se llevan a cabo para la ejecución normal del caso de uso especificado.
- **Postcondiciones:** estado posterior a la ejecución del caso de uso.
- **Excepciones:** casos alternativos que pueden suceder durante la ejecución normal del caso de uso.

Los casos de uso que se han identificado se han clasificado en **tres grupos** teniendo en cuenta quién lleva a cabo la acción: casos de uso desde el punto de vista del **administrador**, desde el punto de vista de un **usuario normal** y propios del **gestor de secretos**.

3.2.1 Casos de uso del administrador

Los cuadros 3.31 a 3.45 detallan los casos de uso que puede llevar a cabo un usuario con el rol de administrador.

CU-01	
Caso de uso	Crear usuario
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de creación de usuario especificando *login* del nuevo usuario, nombre completo del nuevo usuario e indica el archivo que contiene la clave pública del usuario. 2. Se crea correctamente el nuevo usuario.
Postcondiciones	Hay un nuevo usuario registrado en el gestor.
Excepciones	En caso de que el usuario ya exista, se notifica. Si el archivo de la clave es incorrecto, se notifica.

Cuadro 3.31: CU-01, crear usuario.

CU-02	
Caso de uso	Eliminar usuario
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de eliminación de usuario especificando *login* del usuario a eliminar. 2. Se elimina correctamente el usuario especificado.
Postcondiciones	Hay un usuario menos registrado en el gestor.
Excepciones	En caso de que el usuario no exista, se notifica.

Cuadro 3.32: CU-02, eliminar usuario.

CU-03	
Caso de uso	Crear grupo
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de creación de grupo especificando nombre del nuevo grupo, descripción y de forma opcional se indican los usuarios a añadir al grupo. 2. Se crea correctamente el nuevo grupo con los miembros especificados.
Postcondiciones	Hay un nuevo grupo registrado en el gestor.
Excepciones	En caso de que el grupo ya exista, se notifica. Si los usuarios a añadir como miembros son incorrectos, se notifica.

Cuadro 3.33: CU-03, crear grupo.

CU-04	
Caso de uso	Añadir usuarios a un grupo
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de añadir miembros a un grupo especificando nombre del grupo y los usuarios a añadir. 2. Se asignan correctamente los usuarios al grupo especificado.
Postcondiciones	Los usuarios del grupo están actualizados.
Excepciones	En caso de que el grupo no exista, se notifica. Si algún usuario ya pertenece al grupo, se notifica. Si los usuarios a añadir como miembros son incorrectos, se notifica.

Cuadro 3.34: CU-04, añadir usuarios a un grupo.

CU-05	
Caso de uso	Quitar usuarios de un grupo
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de quitar miembros de un grupo especificando nombre del grupo y los usuarios a quitar. 2. Se quitan correctamente los usuarios del grupo especificado.
Postcondiciones	Los usuarios del grupo están actualizados.
Excepciones	En caso de que el grupo no exista, se notifica. Si los usuarios a quitar no existen, se notifica.

Cuadro 3.35: CU-05, quitar usuarios de un grupo.

CU-06	
Caso de uso	Eliminar grupo
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de eliminación de grupo especificando nombre del grupo a eliminar. 2. Se elimina correctamente el grupo especificado.
Postcondiciones	Hay un grupo menos registrado en el gestor.
Excepciones	En caso de que el grupo no exista, se notifica.

Cuadro 3.36: CU-06, eliminar grupo.

CU-07	
Caso de uso	Crear plantilla
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de creación de plantilla especificando nombre de la nueva plantilla y descripción. 2. Se abre una plantilla de ejemplo en el editor por defecto del sistema. 3. Se define el esquema de la plantilla. 4. Se crea la nueva plantilla.
Postcondiciones	Hay una nueva plantilla registrada en el gestor.
Excepciones	En caso de que la plantilla ya exista, se notifica. Si no se dispone de un editor soportado, se notifica. Si hay algún error en el formato de la plantilla se permite al usuario corregirla volviendo a abrir el editor.

Cuadro 3.37: CU-07, crear plantilla.

CU-08	
Caso de uso	Editar plantilla
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de edición de plantilla especificando nombre la plantilla a editar. 2. Se abre la plantilla en el editor por defecto y se edita. 3. Se guarda la plantilla editada.
Postcondiciones	La plantilla especificada ha sido actualizada.
Excepciones	En caso de que la plantilla no exista, se notifica. Si no se dispone de un editor soportado, se notifica. Si hay algún error en el formato de la plantilla se permite corregirla volviendo a abrir el editor.

Cuadro 3.38: CU-08, editar plantilla.

CU-09	
Caso de uso	Asignar plantilla
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de asignación de plantilla especificando el nombre de la plantilla y la carpeta a la que se le va a asignar. 2. Se asigna la plantilla a la carpeta especificada.
Postcondiciones	La carpeta especificada pasa a tener asociada la plantilla especificada.
Excepciones	En caso de que la plantilla no exista, se notifica. En caso de que la carpeta no exista, se notifica. Si la carpeta tiene ya asociada una plantilla, esta es reemplazada por la nueva.

Cuadro 3.39: CU-09, asignar plantilla.

CU-10	
Caso de uso	Eliminar plantilla
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de eliminación de plantilla especificando el nombre de la plantilla a eliminar. 2. Se elimina correctamente la plantilla especificada.
Postcondiciones	El gestor tiene una plantilla menos y las carpetas que la tenían asignada pasan a no tener una plantillas asociada.
Excepciones	En caso de que la plantilla no exista, se notifica.

Cuadro 3.40: CU-10, eliminar plantilla.

CU-11	
Caso de uso	Asignar permisos
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de asignación de permisos especificando el nombre del recurso al que se le van a asignar los permisos y los permisos a asignar. 2. Se actualizan correctamente los permisos del recurso especificado sobre-escribiendo los actuales.
Postcondiciones	Los permisos del recurso especificado están actualizados.
Excepciones	En caso de que el recurso no exista, se notifica. Si algún permiso de los que se quieren asignar no es correcto se notifica y se continua con el resto.

Cuadro 3.41: CU-11, asignar permisos.

CU-12	
Caso de uso	Obtener los permisos de un recurso
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de obtención de permisos especificando el nombre del recurso del que quieren obtener los permisos. 2. Se devuelven los permisos del recurso.
Postcondiciones	Ninguna.
Excepciones	En caso de que el recurso no exista, se notifica.

Cuadro 3.42: CU-12, obtener los permisos de un recurso.

CU-13	
Caso de uso	Listar usuarios
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de listado de usuarios. 2. Se devuelve el listado de los usuarios registrados en el gestor. 2.1. Si no se especifican usuarios, se listan todos los existentes y su información. 2.2. Si se especifican algunos usuarios, se listan solo los usuarios especificados junto con su información.
Postcondiciones	Listado de los usuarios existentes en el gestor.
Excepciones	Ninguna.

Cuadro 3.43: CU-13, listar usuarios.

CU-14	
Caso de uso	Listar grupos
Actores	Administrador del gestor
Precondiciones	El administrador tiene que estar conectado al gestor
Secuencia normal	1. Se ejecuta el comando de listado de grupos. 2. Se devuelve el listado de los grupos registrados en el gestor. 2.1. Si no se especifican grupos, se listan todos los existentes y su información. 2.2. Si se especifican algunos grupos, se listan solo los grupos especificados junto con su información.
Postcondiciones	Listado de los grupos existentes en el gestor.
Excepciones	Ninguna.

Cuadro 3.44: CU-14, listar grupos.

CU-15	
Caso de uso	Listar plantillas
Actores	Administrador del sistema o usuario normal
Precondiciones	El administrador tiene que estar conectado al gestor / El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de listado de plantillas. 2. Se listan las plantillas registradas en el gestor.
Postcondiciones	Listado de las plantillas.
Excepciones	Ninguna.

Cuadro 3.45: CU-15, listar plantillas.

3.2.2 Casos de uso del usuario normal

Los cuadros 3.46 a 3.52 detallan los casos de uso que puede llevar a cabo un usuario normal del gestor. Todos estos casos de uso llevan implícita la autenticación del usuario.

CU-16	
Caso de uso	Listar secretos
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de listado de secretos. 2. Se listan los secretos a los que tiene acceso el usuario. 2.1. Si no se especifica ninguna carpeta, se listan todos los secretos. 2.2. Si se especifica alguna carpeta, se listan solo los secretos que hay dentro de la carpeta especificada.
Postcondiciones	Listado de los secretos.
Excepciones	Ninguna.

Cuadro 3.46: CU-16, listar secretos.

CU-17	
Caso de uso	Crear carpeta
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de creación de carpetas especificando el nombre de la carpeta a crear. 2. Se crea correctamente la carpeta.
Postcondiciones	Nueva carpeta.
Excepciones	Si la carpeta ya existe, se notifica.

Cuadro 3.47: CU-17, crear carpeta.

CU-18	
Caso de uso	Crear secreto
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de creación de secretos especificando el nombre del secreto a crear. 2. Se crea correctamente el secreto. 2.1. Si está bajo el dominio de una plantilla se abre el editor con la plantilla a rellenar por el usuario y se crea. 2.2. Si no está bajo el dominio de una plantilla, se abre el editor vacío y tras completarlo se crea. 2.3. Si se especifica la opción de subir un archivo, se especifica el archivo a subir y se crea.
Postcondiciones	Nuevo secreto.
Excepciones	Si el secreto ya existe, se notifica. Si se intenta subir un archivo en una carpeta que esta bajo el dominio de una plantilla, se notifica y no se crea.

Cuadro 3.48: CU-18, crear secreto.

CU-19	
Caso de uso	Editar secreto
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de edición de secretos especificando el secreto a editar. 2. Se abre el secreto en el editor por defecto del sistema. 3. Se edita la secreto. 4. Se guarda el secreto editado.
Postcondiciones	El secreto especificado ha sido actualizado.
Excepciones	En caso de que el secreto no exista, se notifica. Si no se dispone de un editor soportado, se notifica. Si hay algún error en el formato del secreto se permite al usuario corregirlo volviendo a abrir el editor. Si el secreto que se quiere editar es un archivo, se notifica y no se edita.

Cuadro 3.49: CU-19, editar secreto.

CU-20	
Caso de uso	Obtener un secreto
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de obtención de secretos especificando el nombre del secreto a obtener. 2. Se obtiene el secreto. 2.1. Si no se especifica la opción de guardarlo en un archivo, se muestra en el terminal el contenido del secreto. 2.3. Si se especifica la opción de guardarlo en un archivo, se recupera y se guarda en el archivo especificado.
Postcondiciones	Nuevo secreto.
Excepciones	Si el secreto no existe, se notifica. Si se intenta obtener un archivo que es un archivo sin especificar que se quiere volcar a un archivo, se notifica y no se recupera.

Cuadro 3.50: CU-20, obtener un secreto.

CU-21	
Caso de uso	Eliminar una carpeta
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de eliminación de carpetas especificando el nombre de la carpeta a eliminar. 1.1. Si se especifica eliminación asíncrona, la carpeta y todo su contenido se elimina de forma asíncrona. 2. Se elimina correctamente la carpeta.
Postcondiciones	Ninguna.
Excepciones	Si la carpeta no existe, se notifica. Solo se elimina si se tiene permiso en todos los secretos dentro de la carpeta.

Cuadro 3.51: CU-21, eliminar una carpeta.

CU-22	
Caso de uso	Eliminar un secreto
Actores	Usuario normal
Precondiciones	El usuario debe estar registrado en el gestor
Secuencia normal	1. Se ejecuta el comando de eliminación de secretos especificando el nombre del secreto a eliminar. 1.1. Si se especifica eliminación asíncrona, el secreto se elimina de forma asíncrona. 2. Se elimina correctamente el secreto.
Postcondiciones	Ninguna.
Excepciones	Si el secreto no existe, se notifica. Si se intenta eliminar un secreto al que no se tiene acceso, se notifica y no se elimina.

Cuadro 3.52: CU-22, eliminar un secreto.

3.2.3 Casos de uso del gestor de secretos

Los cuadros 3.53 a 3.55 detallan los casos de uso que puede llevar a cabo el gestor sin intervención humana.

CU-23	
Caso de uso	Eliminar archivos basura
Actores	Gestor de secretos
Precondiciones	Ninguna.
Secuencia normal	1. El gestor ejecuta una tarea que comprueba si existen archivos de secretos que no están registrados en la BD del gestor. 2. Los archivos que no estén en la BD se eliminan.
Postcondiciones	El gestor libera espacio ocupado.
Excepciones	Ninguna.

Cuadro 3.53: CU-23, eliminar archivos basura.

CU-24	
Caso de uso	Comprobar la seguridad de una contraseña
Actores	Gestor de secretos
Precondiciones	Ninguna.
Secuencia normal	1. El gestor ejecuta una tarea que comprueba para aquellos secretos que contengan una contraseña si esta es lo suficientemente segura. 2. Si no es segura, se marca como insegura.
Postcondiciones	Ninguna.
Excepciones	Ninguna.

Cuadro 3.54: CU-24, comprobar la seguridad de una contraseña.

CU-25	
Caso de uso	Comprobar si un dominio ha sufrido una brecha de seguridad
Actores	Gestor de secretos
Precondiciones	Tener conexión a internet.
Secuencia normal	1. El gestor ejecuta una tarea que comprueba para aquellos secretos que contengan un dominio si este ha sufrido una brecha de seguridad recientemente. 2. Si ha sufrido alguna brecha, se marca como insegura.
Postcondiciones	Ninguna.
Excepciones	Si no hay conexión a internet no se hace nada.

Cuadro 3.55: CU-25, comprobar si un dominio ha sufrido una brecha de seguridad.

3.3 Análisis de las posibles soluciones

Teniendo en cuenta el análisis de requisitos previo, se ha pensado en múltiples soluciones que cubrieran los requisitos especificados.

La primera solución planteada consistía en la **estructuración de los secretos** según la misma aproximación que Password-Store [11], usando **Git** como mecanismo de distribución. Como aspectos positivos cabe destacar que simplifica el problema puesto que no requiere un gran despliegue de infraestructura, pero en cuanto a aspectos negativos, nos encontramos con la necesidad de realizar la recompilación de la estructura de secretos tras actualizar los permisos de los usuarios sobre los mismos y el inconveniente de que la estructura de secretos completa puede ser vista por todos los usuarios. Esta alternativa ha sido considerada pero finalmente se ha descartado debido a los inconvenientes citados.

Otra solución posible habría sido la utilización de un **sistema de cifrado distinto a GPG** como podría ser el uso de criptografía simétrica con AES-256 o *Blowfish* pero, debido al re-

quisito R-27, autenticación mediante un *token* y uso de GPG, que obliga al uso de GPG como sistema de cifrado, ha quedado totalmente descartada.

Por último se ha considerado utilizar una **arquitectura cliente-servidor** en la que el servidor almacena todos los secretos y los cifra con su clave pública, manteniendo así el uso de GPG como sistema de cifrado, y el cliente permite a los usuarios interactuar con el servidor introduciendo una capa de autenticación basada en GPG como se especifica en los requisitos. A continuación se han listado los aspectos positivos y negativos de esta solución en comparación con la estructura que sigue la herramienta actual:

- Aspectos positivos:
  - Centralización del control de los recursos.
  - Protección de los datos que maneja el gestor del acceso a los mismos por parte de los programas del lado cliente.
  - Acceso al servidor solo por parte de usuarios autorizados.
- Aspectos negativos:
  - Añade mayor carga al tráfico de red.
  - Añade un SPOF puesto que si el servidor está caído se pierde el acceso completo a los datos.

De las soluciones mencionadas en la lista anterior, se ha decidido llevar acabo la última citada puesto que es la que mejor cumple los requisitos especificados. Por este motivo, y debido al entorno en el que se usará la aplicación, la arquitectura cliente-servidor clásica resuelve perfectamente sus necesidades.

3.4 Solución propuesta

La arquitectura escogida para el gestor de secretos ha sido una arquitectura cliente-servidor en la cual el servidor se encarga de toda la lógica de gestión de secretos, almacenamiento de los mismos, ejecución de tareas asíncronas, realización de comprobaciones y demás tareas explicadas más adelante con mayor detalle. Por otro lado, el cliente se comunica con el servidor mediante llamadas al API que este expone con las funcionalidades que puede realizar un usuario normal.

De forma general el gestor consta de **dos aplicaciones**: la aplicación que corre en el servidor y se encarga de toda la lógica de gestión de secretos, así como los servicios necesarios para las funcionalidades que se quieren soportar (p.e. la base de datos MySQL), y otra aplicación que corre en cada uno de los clientes que utilicen el servidor y que solo interactúa con el

servidor haciendo peticiones al API que este expone, realizando tareas simples de descifrado y formateado de datos.

El servidor tendrá su propio par de claves GPG (clave pública y clave privada), tanto para mantener los secretos almacenados cifrados en todo momento, como para poder crear el sistema de autenticación de los usuarios. Además, el servidor también almacenará todas las claves públicas de los usuarios que se vayan creando en el mismo, imprescindible para la implementación del flujo de autenticación. Para lograr la autenticación, el servidor genera un *token* nuevo para cada usuario que se asocia a este y que es el valor que se envía cifrado entre cliente y servidor para realizar la autenticación (el proceso de autenticación ha sido explicado con más detalle en la sección 4, página 43).

Por otra parte, **el cliente** solo necesita tener configurado un entorno GPG así como su propio par de claves para que el administrador lo pueda registrar en el servidor y así poder interactuar con el mismo.

Debido a la existencia de un servidor en el que se almacenan todos los datos se han creado funcionalidades orientadas al papel de **administrador**, denominadas funcionalidades de administración, y otras orientadas al **usuario normal**, denominadas funcionalidades de usuario. Debido a esta división en cliente-servidor y la división de las tareas, podemos diferenciar dos grandes roles a la hora de utilizar el gestor:

- **Administrador:** tiene acceso directo al servidor en el que se ejecuta el gestor de secretos y que puede realizar las tareas detalladas en la sección 3.2.1.

  Además de todas las funcionalidades de administración que ofrece el gestor de secretos, el administrador tiene control total sobre el resto de servicios que corren sobre el sistema en el que está instalado el servidor, así como los demás servicios necesarios como BBDD, servicio para ejecución de tareas asíncronas y servicio para tareas periódicas.

- **Usuario normal:** utiliza el gestor de secretos mediante el uso de la aplicación cliente instalada en su equipo, puede realizar las tareas típicas de un usuario normal detalladas en la sección 3.2.2.

  Además, a todas las tareas que puede realizar hay que añadir que en el caso del listado, solo puede ver aquellos recursos sobre los que se la han concedido permisos. De igual forma, solo podrá crear carpetas y secretos en las ubicaciones en las que tenga los permisos necesarios y solo podrá eliminar aquellos recursos a los que tenga acceso.

El **soporte para OTPs** se ofrece mediante la creación de una carpeta que tenga asociada una plantilla con soporte para OTPs permitiendo al cliente crear secretos almacenando en ellos los datos necesarios para la generación de la OTP y cuando recupere el secreto, obtendrá el valor de la OTP correspondiente.

De cara a la implementación los **principales problemas** que se han generado debido a la elección de esta aproximación, se centran mayoritariamente en las librerías disponibles para, por ejemplo, todo el proceso relacionado con el uso de GPG, soporte para OTPs, manipulación de formatos que se han utilizado como JSON o YAML. Otra parte que ha resultado algo compleja ha sido la utilización de herramientas para la ejecución de tareas asíncronas y tareas periódicas y su integración con el servidor.

3.5 Análisis de seguridad

Desde el punto de vista de la seguridad, se ha de destacar la existencia de un SPOF que aparece como consecuencia del almacenamiento de los secretos en el servidor usando la clave asimétrica de éste. Si alguien con malas intenciones consiguiera acceder al servidor, tendría acceso a todos los secretos allí almacenados.

A pesar de la existencia de este SPOF, esta es la mejor de las soluciones posibles. Este problema no es único de esta aplicación sino que es común en las arquitecturas cliente-servidor que por un motivo u otro necesitan limitar el acceso a un recurso utilizando algún tipo de clave. Existen diversas estrategias para mitigarlo: implementar el servicio de tal forma que al iniciarlo solicite una clave que permita desbloquear el acceso al recurso, usar un módulo hardware externo para almacenar la clave, integrar el servidor con servicios como AWS Key Management o Oracle Vault, etc. Algunas de estas estrategias han sido analizadas con más detalle en el capítulo 5.

Por otra parte, la seguridad del gestor de secretos se basa en el uso de criptografía asimétrica GPG como base para la autenticación de usuarios y el cifrado de la información, obteniendo un buen nivel de protección de la información. La capa de autenticación y cifrado basada en GPG ha sido creada con el objetivo de añadir un nivel extra de seguridad aprovechando la infraestructura GPG utilizada por la aplicación, securizando todas las comunicaciones entre el cliente y el servidor. Con el objetivo de aumentar la seguridad en las comunicaciones, se utiliza una capa TLS entre el cliente y el servidor que añade un nivel extra de protección.

Capítulo 4

Diseño de la solución

En este capítulo se ha explicado en detalle el diseño de la solución propuesta. En primer lugar se ha realizado un análisis de la arquitectura empleada explicando todos los elementos que la componen y la funcionalidad de cada uno. También se han detallado el modelo de datos empleado y la forma en la que se han cubierto todos los requisitos listados en la sección 3.1, página 19. Por último, se ha realizado un breve análisis de las librerías y herramientas utilizadas en el desarrollo de la aplicación.

4.1 Arquitectura software

Como ya se ha explicado en la sección 3.4, página 39, se ha empleado una arquitectura cliente-servidor para la creación del gestor de secretos. El gestor se ha descompuesto en dos componentes principales: la aplicación que corre en el lado servidor y la que corre en el lado cliente. En la figura 4.1 se muestra un diagrama que ilustra la arquitectura completa de la aplicación con todos sus componentes.

En torno al servidor hay todo un conjunto de componentes que son necesarios para el funcionamiento completo del sistema. La aplicación del servidor **mrsecrets-server** ha sido desarrollada en Python, más concretamente en su versión 3.6, con la ayuda del *framework* de desarrollo Django. Se ha escogido este *framework* debido a las ventajas que ofrece como la integración con Redis [23], soporte para múltiples bases de datos (p.e. MySQL, SQLite, PostgreSQL, etc), fácil integración con sistemas de ejecución de tareas asíncronas como Celery [24], buena integración con sistemas de monitorización como Sentry [25] y demás ventajas que ayudan a un desarrollo más ágil. La aplicación servidor expone un API REST para la interacción de los clientes. A su vez, la aplicación servidor depende y se integra con los componentes siguientes:

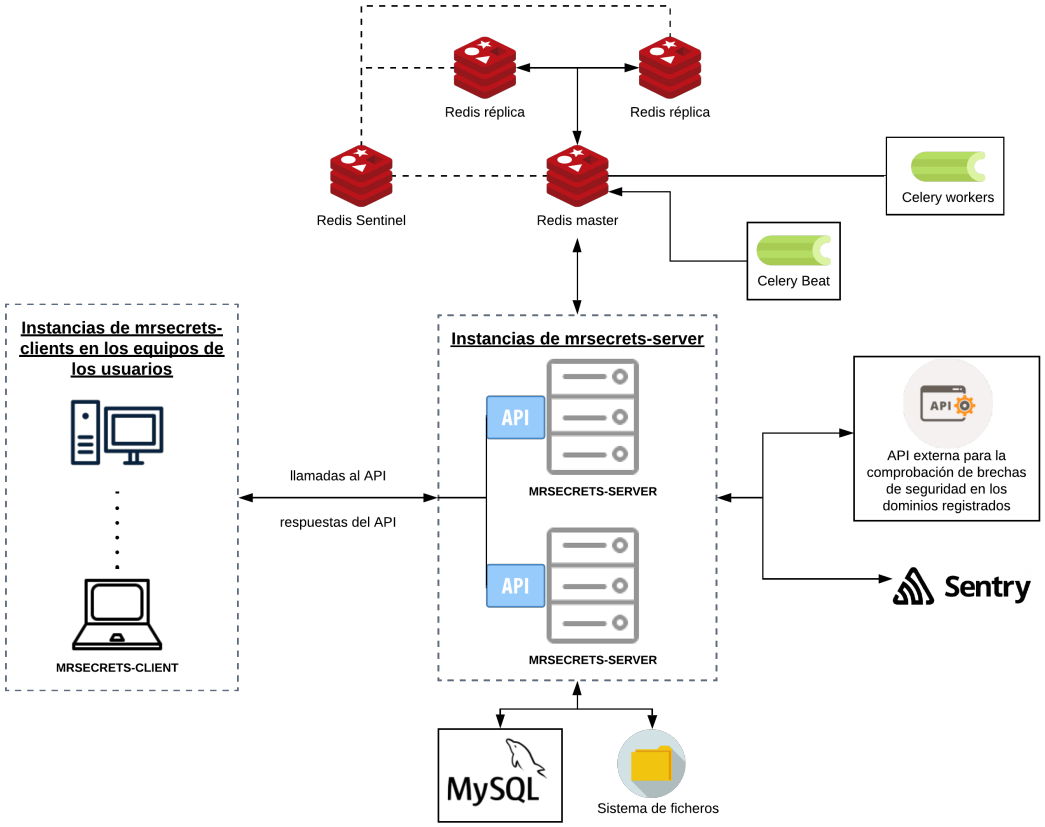

Figura 4.1: Arquitectura

- **Redis**, cuya principal funcionalidad es servir como *broker* para transmitir la ejecución de tareas asíncronas y periódicas del gestor de secretos a Celery (la conexión entre Celery y Redis, se basa en una lectura bloqueante en Redis de una lista en la que se encolan las tareas insertadas por Celery Beat o el propio gestor de secretos).

- **Celery** es un gestor de colas para tareas asíncronas que posteriormente son ejecutadas en segundo plano mediante los Celery *workers*. Los Celery *workers* son procesos que ejecutan las tareas encoladas por Celery y devuelven el resultado de las mismas. Además Celery también ofrece soporte para la ejecución de tareas periódicas mediante el uso de Celery Beat [26]. Para encolar tareas en Celery es necesario utilizar un *broker*, en nuestro caso Redis, que transmite las tareas en forma de mensaje para posteriormente ser encoladas y finalmente ejecutadas por Celery. Cabe destacar que las tareas asíncronas son enviadas por el gestor de secretos, mientras que las periódicas las envía Celery Beat en los instantes de tiempo especificados.

- **MySQL**, para la persistencia del modelo de datos creado en Django.
- **Sistema de ficheros** en el que se almacenan los secretos creados por los usuarios.
- **Sentry**, herramienta de monitorización de aplicaciones web en tiempo real que permite hacer seguimiento de errores en las aplicaciones en entornos de producción.

Por otro lado, la aplicación cliente denominada **mrsecrets-client** será instalada en todos los equipos que vayan a utilizar este gestor. Al igual que el servidor, ha sido desarrollado en Python, en su version 3.6, junto con el uso del *framework* de desarrollo Django. Cabe destacar que a esta aplicación le basta con disponer de una infraestructura GPG instalada y configurada. Por lo demás la aplicación esta compuesta principalmente por comandos Django que encapsulan llamadas al API y tareas básicas de cifrado y descifrado.

Finalmente, en cuanto a la interacción entre los componentes cliente (**mrsecrets-client**) y servidor (**mrsecrets-server**), el diagrama de secuencia de la figura 4.2 muestra el proceso de obtención de un secreto del gestor. Además, este diagrama ilustra el funcionamiento del proceso de autenticación. Cabe destacar que el proceso de autenticación no sigue el esquema de la aplicación web tradicional. No existen conceptos como *login*, sesión y *logout*, sino que la autenticación es realizada con cada petición del cliente al servidor mediante el uso de un *token* firmado con las claves GPG correspondientes.

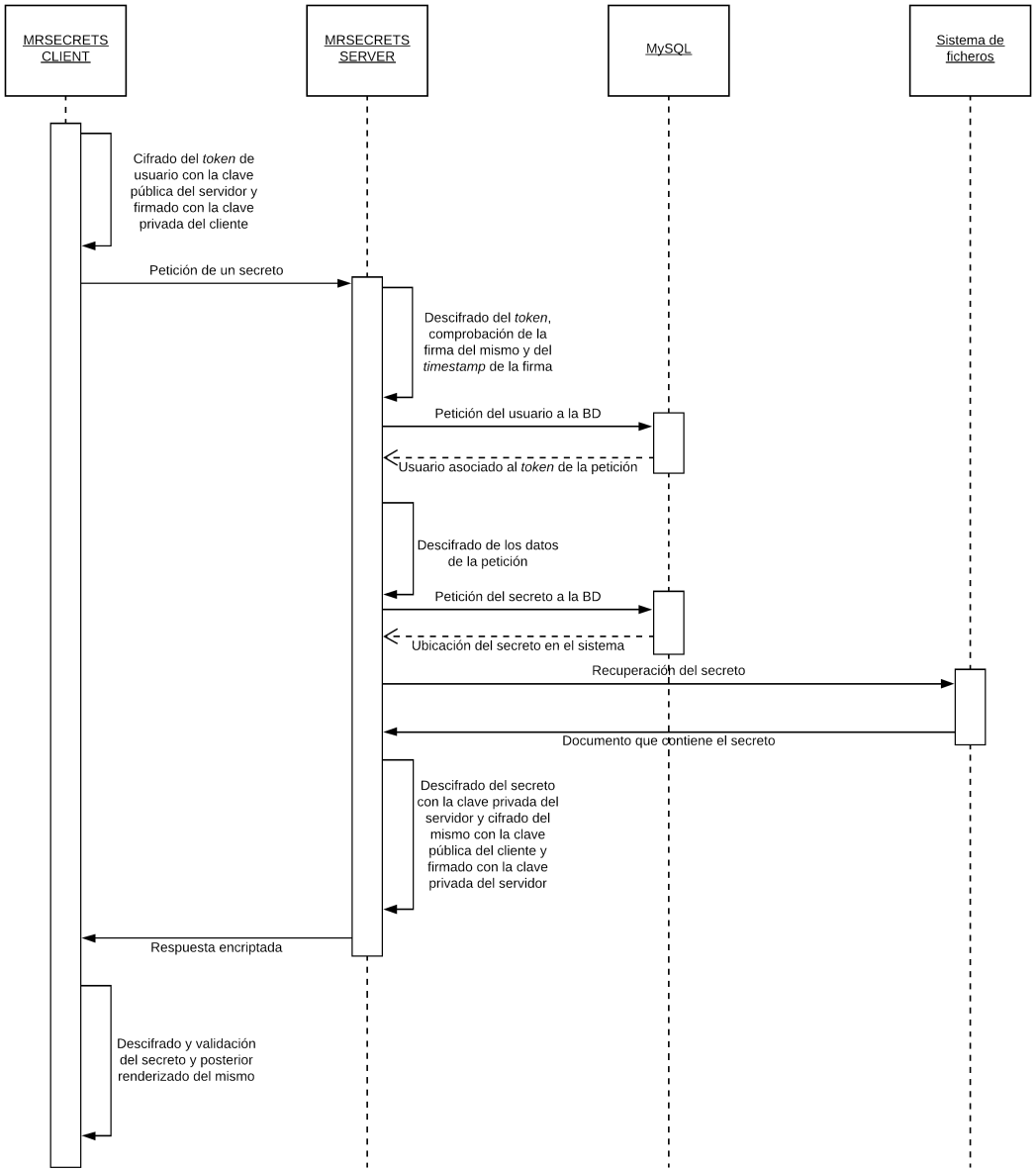

Figura 4.2: Diagrama de secuencia del proceso de obtención de un secreto

4.1.1 Alta disponibilidad

En los sistemas que requieren estar accesibles en todo momento debido a la importancia de los datos que gestionan, es fundamental implementar mecanismos de alta disponibilidad. A continuación se han descrito con detalle las técnicas empleadas para dar soporte de alta disponibilidad a los componentes que conforman la arquitectura de la aplicación servidor.

La alta disponibilidad en el **servidor**, entendiendo como servidor el propio **mrsecrets-**

server, se logra mediante escalado horizontal, es decir simplemente replicando la aplicación Django. Por otro lado, a continuación se detalla la estrategia seguida para proveer de alta disponibilidad a los componentes de los que depende mrsecrets-server.

Con el objetivo de proporcionar alta disponibilidad en **Redis**, su configuración consta de un nodo *master* y una o más réplicas [27] que se mantienen sincronizadas con *master* y, a su vez, los nodos están monitorizados mediante el uso de Redis Sentinel [28]. **Redis Sentinel** es una de las soluciónes de alta disponibilidad que ofrece Redis y que, mediante un sistema de *failover*, se encarga de monitorizar los nodos Redis (*master* y réplicas) y en caso de detectar algún fallo en el *master*, restaura el sistema convirtiendo una réplica en *master* y reconfigurando las demás de tal modo que su *master* sea actualizado. En esta aplicación, se ha implementado el soporte para su compatibilidad con la alta disponibilidad de Redis. Fundamentalmente consiste en que el servidor, mrsecrets-server, utiliza Sentinel para en cada petición HTTP, obtener el nodo Redis que actúa como maestro y así saber que nodo consultar.

Dar soporte de alta disponibilidad a la capa **Celery** se reduce a replicar el servicio. A su vez, para tolerar fallos en la capa Redis, Celery se integra con Redis Server y con Redis Sentinel de forma similar a mrsecrets-server. Por otro lado, **Celery Beat** no permite la implementación de alta disponibilidad, cuestión sin importancia por su naturaleza dentro de la arquitectura global.

La alta disponibilidad en **MySQL** se logra mediante el uso de la tecnología *High-Availability MySQL cluster* [29], basada en el despliegue de al menos dos instancias MySQL en configuración activo-pasivo o activo-activo.

En cuanto al **sistema de ficheros** donde se almacenan los secretos, el soporte de alta disponibilidad pasaría por el uso de un sistema de ficheros en *cluster* como por ejemplo GlusterFS [30].

Por último, sobre **Sentry** no se puede aplicar alta disponibilidad puesto que se trata de un servicio externo que proporciona sentry.io. Del mismo modo que Celery Beat, este es un elemento secundario dentro de la arquitectura y las consecuencias de que temporalmente no esté disponible son menores.

4.1.2 Persistencia

En el sistema se pueden diferenciar **tres tipos de persistencia**: base de datos relacional, sistemas de ficheros y la base de datos clave-valor Redis.

La **base de datos relacional** empleada por el lado servidor del gestor para la persistencia de las entidades que componen la aplicación, es **MySQL**. Aunque se ha decidido emplear MySQL por su popularidad en el mundo *open-source*, gracias a que toda la construcción de la aplicación se ha apoyado en Django, también están soportadas de forma nativa otras bases de datos como PostgreSQL y SQLite.

El **sistema de ficheros** es empleado para almacenar el anillo GPG que contiene las claves utilizadas por la aplicación y para el almacenamiento de los secretos que se suben al servidor y que quedan registrados en la base de datos MySQL.

La base de datos **Redis** se utiliza con los siguientes fines: funciona como una caché de nivel de aplicación para el servidor, actúa como *broker* para el gestor de colas Celery y almacena los resultados de Celery. En este caso al igual que con MySQL existen alternativas a Celery para gestionar colas de tareas como por ejemplo RabbitMQ que se podrían utilizar con unos pequeños ajustes en la configuración de Celery.

4.1.3 Modelo de datos

La figura 4.3, ilustra un diagrama EER (*Enhanced Entity-Relationship*) en el que se muestran las conexiones entre las entidades que conforman la estructura de datos que utiliza el gestor de secretos. Existen cinco tipos de entidades principales:

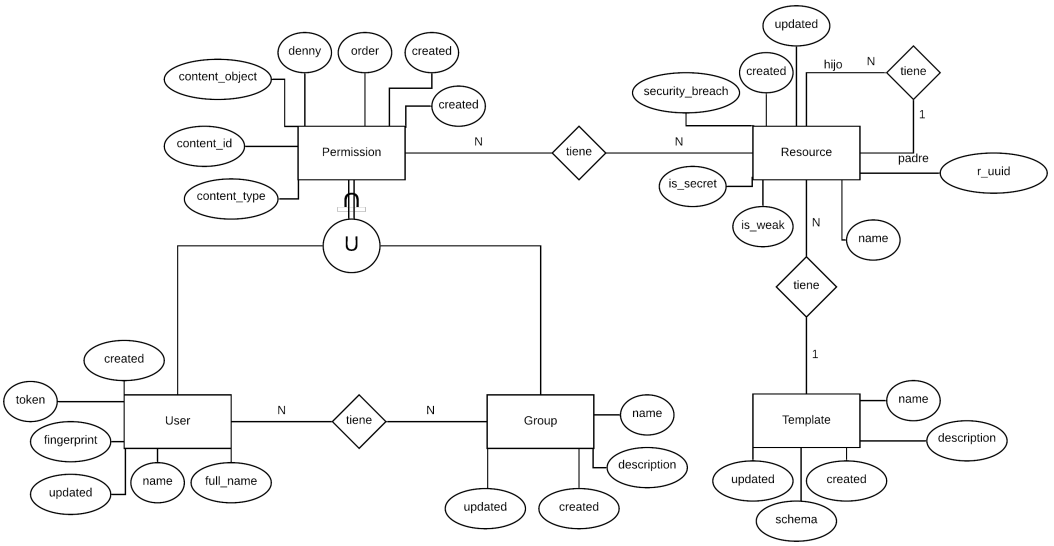

Figura 4.3: Modelo de datos

- ***user*:** entidad que almacena todos los usuarios dados de alta en el gestor de secretos. Cada vez que se registra un nuevo usuario es necesario especificar nombre, nombre completo y un archivo que contenga la clave pública del usuario. En el proceso de creación de usuario el sistema genera un *token* que deberá proporcionarse al usuario correspondiente.
- ***group*:** entidad que almacena todos los grupos creados en el gestor de secretos. Permite agrupar a los usuarios facilitando el proceso de gestión de permisos.

- ***template*:** entidad que almacena las plantillas utilizadas para definir un esquema común para los secretos que se almacenan en una carpeta determinada. El concepto de ***template*** se basa en la utilización de JSON Schemas que permiten definir una serie de campos que deberán contener los secretos que se encuentren bajo su dominio.

- ***resource*:** entidad que almacena una referencia a los recursos, carpetas o secretos, que se guardan en el gestor de secretos.

- ***permissions*:** entidad que permite determinar que usuarios o grupos pueden acceder a un determinado recurso.

Los permisos que se asignan a los recursos pueden ser tanto a nivel de grupo como de usuario, por lo que el tipo de entidad *permissions* se forma a través de una unión total con *user* y *group*. Por otra parte, un *resource* puede tener asociados varios *permissions*, en los cuales es relevante el orden en el que se especifican.

El sistema de permisos se basa en expresiones compuestas por usuarios y/o grupos. A su vez, tanto los usuarios como los grupos pueden llevar delante un símbolo ("+" o "-") que indica si el usuario o grupos al que van ligados tienen o no permiso de acceso al recurso. El análisis de permisos se hace recorriendo la estructura del árbol de directorios registrado en la base de datos comenzando por la raíz hasta llegar al recurso deseado.

Teniendo en cuenta la figura 4.4, si el usuario bob quiere recuperar el recurso /ALL/dev/tfg, se analizarán los permisos de la siguiente forma:

1. ALL: +guillermo, +carlos

2. dev: +guillermo, +carlos, +moises, +bob

3. tfg: +guillermo, +carlos, +moises

Por lo tanto, bob no tendrá permisos en el recurso tfg. De la misma manera, si moises quiere acceder a /ALL/sys/saltstack no podrá puesto que el resultado será: +guillermo, +carlos.

Además, por encima de todos los grupos existe un grupo especial denominado ***keepers***. Los usuarios que pertenecen al grupo ***keepers*** tienen acceso a todos los recursos almacenados. Antes de comprobar si un usuario tiene acceso a un recurso determinado, se comprueba si pertenece al grupo ***keepers*** y en caso afirmativo se le devuelve el recurso y en caso negativo, se realiza el análisis de permisos de forma normal.

Para toda la interacción con la base de datos se ha utilizado el **ORM (Object-Relational Mapping) de Django**. Esencialmente, el **ORM** se encarga de realizar toda la comunicación con la base de datos abstrayendo de esa complejidad al desarrollador.

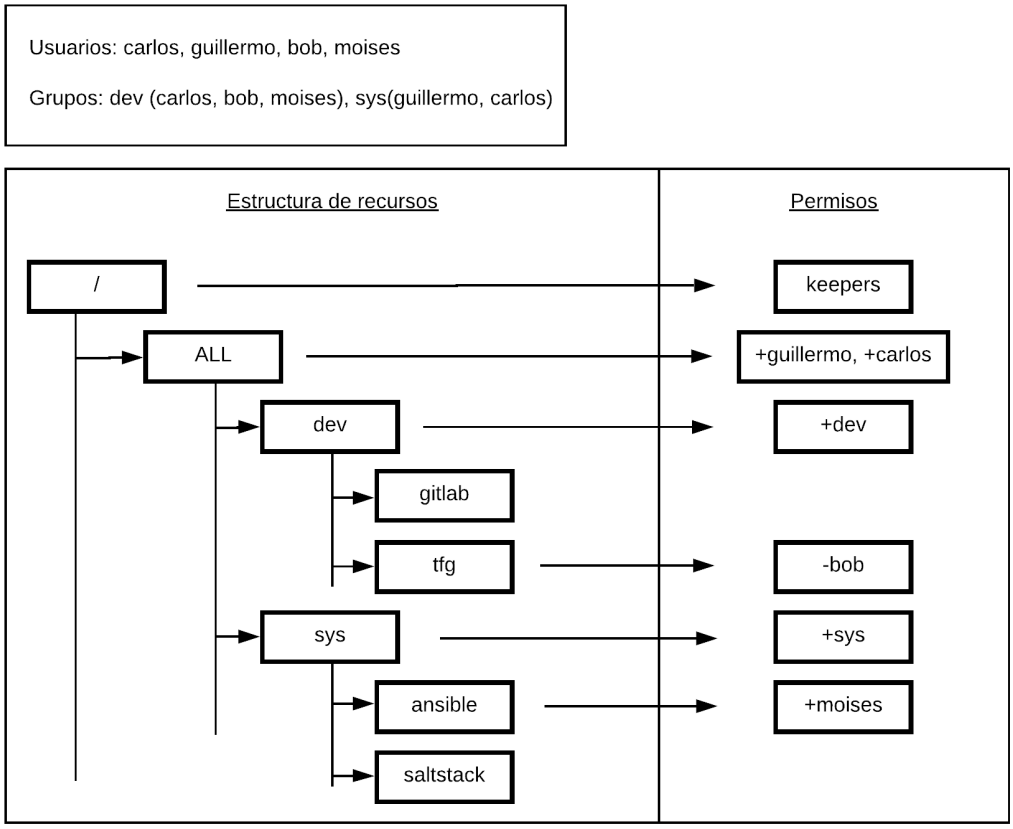

Figura 4.4: Estructura de recursos de ejemplo

4.1.4 Ejecución asíncrona de tareas

La ejecución de tareas asíncronas es una parte importante en el gestor de secretos que permite la ejecución de tareas de revisión de secretos para poder realizar comprobaciones de seguridad, así como realizar tareas en segundo plano evitando bloquear los procesos HTTP con tareas muy costosas en tiempo.

Las tareas asíncronas del gestor son: comprobación de dominios que hayan sufrido brechas de seguridad recientemente, comprobación de la resistencia y la seguridad de las contraseñas, eliminación de archivos del sistema de ficheros no referenciados por el modelo de datos y eliminación de recursos en cascada.

4.2 Análisis de librerías y herramientas utilizadas

4.2.1 Herramientas generales

Como ya se adelantó en la sección 4.1, tanto la parte cliente como la parte servidora se han implementado en el lenguaje de programación Python, en su versión 3.6, junto con el *framework* de desarrollo Django [8]. Django es un *framework* de desarrollo web que permite la creación de aplicaciones abstrayendo al programador de determinadas complejidades del desarrollo web agilizando el proceso de creación de aplicaciones. Uno de sus puntos fuertes es la gran cantidad de documentación disponible que es constantemente actualizada.

Se ha utilizado Git como herramienta de control de versionado para un mejor seguimiento del desarrollo y de los cambios y añadidos que se han ido realizando sobre la aplicación. Puesto que el proyecto todavía se encuentra en una fase de desarrollo temprana se ha decidido utilizar un repositorio privado en Bitbucket [31].

La librería empleada para la integración con GPG ha sido *python-gnupg* [32]. Esta librería es muy sencilla y lo único que hace es realizar llamadas al binario del programa GPG instalado en el sistema. Como se ha mencionado en el capítulo 7, el reemplazo de esta librería es uno de los puntos a mejorar en el futuro.

4.2.2 Herramientas para el despliegue de entornos

Otras herramientas que se han utilizado durante la fase de desarrollo del proyecto han sido **VirtualBox** [33], **Vagrant** [34], **Ansible** [35] y los **virtualenvs** de Python. **Vagrant** se ha utilizado como herramienta para la creación y despliegue del entorno de desarrollo utilizado durante todo el proceso, permitiendo la creación de entornos virtuales fácilmente replicables. Junto con Vagrant, y como proveedor de servicio de virtualización, se ha utilizado **VirtualBox** debido a su carácter *open-source* y gran popularidad.

La herramienta que más hay que destacar es **Ansible**, una plataforma de software libre para configurar y administrar máquinas, que puede ser utilizada para los siguientes fines: aprovisionamiento, gestión de la configuración, despliegue de aplicaciones y orquestación entre otras. En este caso se ha utilizado para realizar el aprovisionado de forma automática del entorno de desarrollo. Para esto se ha programado la lógica necesaria en Ansible e integrado con Vagrant. Para evitar contaminar el entorno de desarrollo con paquetes Python y lograr agruparlos en un punto controlado, se han utilizado los **virtualenvs** que proporciona Python para un control más detallado de las librerías así como de la versión de Python utilizada.

Capítulo 5

Implementación

En este capítulo se ha entrado en detalle en temas relacionados con la implementación y con ciertas funcionalidades que se han considerado suficientemente interesantes y relevantes como para ser explicadas con mayor granularidad.

5.1 *Middlewares* y *backends* a medida

Dentro de los conceptos de Django podemos encontrar las palabras ***middleware*** y ***backend* de autenticación**. En el contexto de Django, un ***middleware*** es el punto de la aplicación que se encarga de gestionar las peticiones y las respuestas, permitiendo realizar diferentes acciones sobre las mismas. Por otro lado, el ***backend* de autenticación** permite añadir mecanismos de autenticación diferentes a los habituales.

Para poder implementar autenticación basada en GPG, se ha creado un ***backend* de autenticación a medida** que se encarga de realizar este proceso. El proceso consta fundamentalmente de las siguientes fases: descifrado del *token* recibido en la petición con la clave privada del servicio (mrsecrets-server), obtención del usuario correspondiente al *token* recibido, validación de la firma del *token* utilizando la clave pública del usuario y, finalmente, realizar el descifrado completo del mensaje con la clave privada del servicio.

El punto más interesante a destacar de esta implementación es el uso de un *token* especial que se genera de forma aleatoria para cada usuario y que se ubica en la configuración de cada aplicación cliente. Este *token* es entregado a cada usuario por el administrador una vez este da de alta al nuevo usuario en el gestor de secretos. Además de funcionar como el tradicional *token* de acceso a un API, todo el proceso de autenticación se basa en el uso de este *token* como medio para comprobar firmas, obtener usuario, comprobar validez en el tiempo de la petición, etc.

Además, se ha implementado otro ***middleware*** que se encarga de gestionar las **redirecciones HTTP a HTTPS** permitiendo configurar el uso de una capa TLS directamente desde

la aplicación mrsecrets-server.

5.2 Almacenamiento de claves

Como ya se ha mencionado en la sección 3.5, página 41, la ubicación de la clave privada que utiliza el servicio es un problema de cara a la seguridad del sistema y sus datos. Actualmente, esta clave es almacenada en el anillo de claves GPG del sistema al igual que las claves públicas de los usuarios dados de alta y la clave simétrica para acceder a la misma, forma parte de la configuración del servicio. Desde el punto de vista del empaquetado de la aplicación, durante la ejecución de los *scripts* de pre-instalación y post-instalación, se crearía un usuario específico que sería el único con permisos para acceder al anillo de claves utilizado por la aplicación y al archivo de configuración. Este usuario sería el encargado de correr el servicio, aumentando el nivel de protección de la clave.

Esta es la estrategia habitual para la protección de secretos del lado servidor en aplicaciones con requisitos similares. Teniendo en cuenta la importancia de la información almacenada en el gestor de secretos, existen una serie de estrategias alternativas que podrían llegar a considerarse e integrarse sin demasiada complejidad:

- **Solicitud de la clave simétrica con la que descifrar la clave privada almacenada en el anillo GPG durante el arranque del servicio:** esta estrategia ofrece un alto nivel de seguridad puesto que la clave simétrica solo es guardada en memoria durante la ejecución del servicio. El principal problema es que se requiere interacción de un administrador / usuario que conozca la clave simétrica para iniciar el servicio, eliminando cualquier posibilidad de automatización de la gestión del mismo.

- **AWS Key Management Service [36]:** este servicio ofrecido por Amazon, permite un control centralizado de las claves utilizadas por ejemplo por aplicaciones o servicios de forma segura, permitiendo en este caso almacenar la clave simétrica en un servidor de Amazon y configurar el servidor para que acceda a este para utilizarla. El uso que mejor encaja con el gestor de secretos es realizar una integración de la aplicación con el SDK (*Software Development Kit*) de cifrado de AWS que se integra con AWS KMS y así poder realizar el cifrado dentro de la aplicación servidor utilizando la clave almacenada en los servidores KMS de Amazon.

- **Oracle Database Vault [37]:** al igual que la solución ofrecida por Amazon se basa en el almacenamiento de claves en una ubicación controlada y muy restringida para proteger la clave simétrica en caso de accesos no permitidos a la instancia del servidor. A diferencia de la solución de Amazon, Oracle Vault almacena las claves en una base de

datos con unas opciones de configuración muy detalladas y un gran nivel de granularidad a la hora de gestionar accessos. Además de permitir un gran nivel de detalle a la hora de controlar accesos, es posible especificar factores fuera de la base de datos como dirección IP, métodos de autenticación y nombre de programas, dificultando aún más el acceso a la clave del servicio.

Como ya se ha comentado en la sección 3.5, en esta aplicación se ha decidido almacenar la propia clave simétrica en el servidor en el que se encuentra el servicio, dentro del archivo de configuración puesto que es el punto más protegido y difícil de acceder de la arquitectura.

5.3 Despliegue en AWS

Se ha realizado un despliegue preliminar en un entorno muy próximo al final mediante el uso de AWS. Este despliegue ha servido para probar la herramienta con un grupo seleccionado de usuarios y poder validar el correcto funcionamiento de la misma así como el cumplimiento de los requisitos desde el punto de vista del usuario.

Para acceder al servidor, los usuarios han instalado la aplicación cliente en sus equipos para posteriormente acceder al gestor y utilizarlo con normalidad. Además, este despliegue de la aplicación ha servido para comprobar la compatibilidad con diversos sistemas como Mac, Ubuntu y Debian.

5.4 Planes *open-source*

Como ya se ha comentado en otras secciones, uno de los objetivos finales de este proyecto es convertirlo en una herramienta *open-source* aportando el trabajo y la utilidad del mismo a la comunidad.

Para esto se ha tenido en cuenta una serie de licencias como por ejemplo licencias BSD, licencias Creative Commons y licencias GPL. Finalmente se ha decidido optar por la licencia GPL puesto que es una de las licencias libres con más influencia y peso en la actualidad. Se ha optado por esta porque permite que cualquier usuario pueda utilizar, modificar y distribuir esta aplicación de forma libre y bajo la condición de que todas las variantes que surjan de este proyecto o todos los proyectos que la utilicen, estarán sujetos a la misma licencia.

5.5 Miscelánea

Otros aspectos relevantes de la implementación que se han considerado interesantes, son los siguientes:

- Para una mayor facilidad y una mejor experiencia de usuario a la hora de crear secretos bajo el dominio de una plantilla, como se muestra en la figura 5.1, los formularios a completar con los campos de la plantilla son mostrados en formato YAML en lugar de JSON.

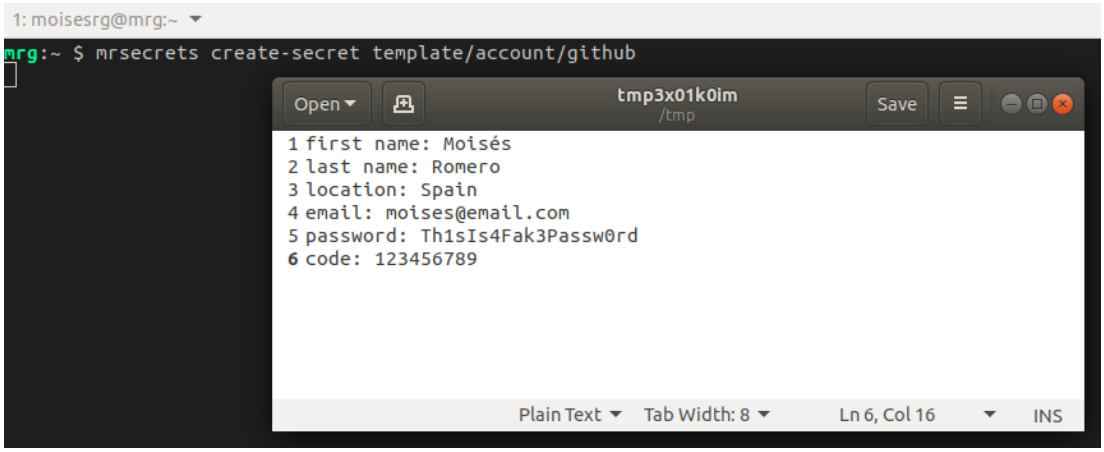

Figura 5.1: Creación de un nuevo secreto bajo un directorio con plantilla asociada

- Si se cambia la plantilla de una carpeta y esta contiene secretos, la próxima vez que se edite algún secreto con el formato antiguo (figura 5.2), los campos comunes entre la plantilla antigua y la nueva, se mantienen y, aquellos que desaparecen, se muestran junto con sus valores avisando de que al guardar serán eliminados.

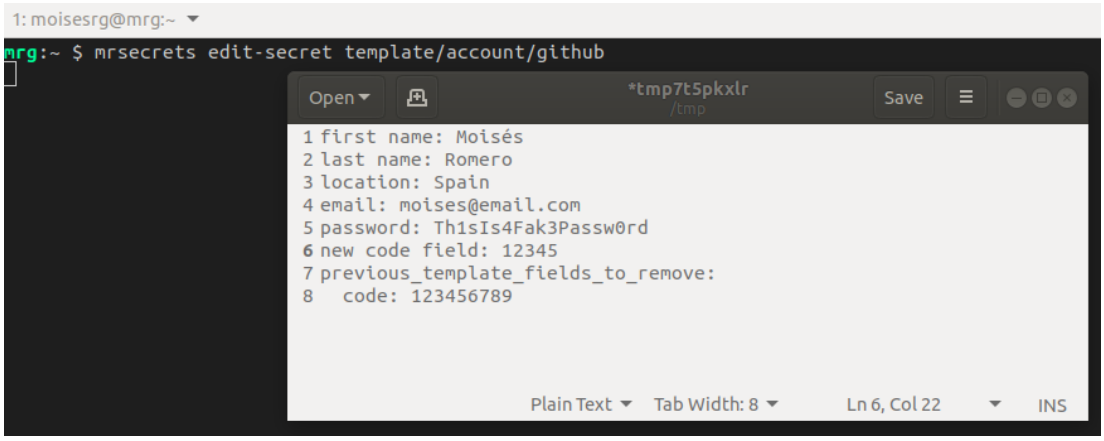

Figura 5.2: Edición de un secreto tras modificar la plantilla de la que depende

- La implementación del proceso de obtención de usuarios, recursos, etc, en el lado servidor, no es todo lo eficiente que podría debido al elevado número de consultas a la base de datos. Esta ineficiencia podría arreglarse mediante la implementación de políticas de

cacheo, aprovechando la infraestructura Redis ya disponible.

- Como se puede apreciar en la figura 5.3, el listado de los recursos incluye una serie de campos que aportan información útil al usuario:

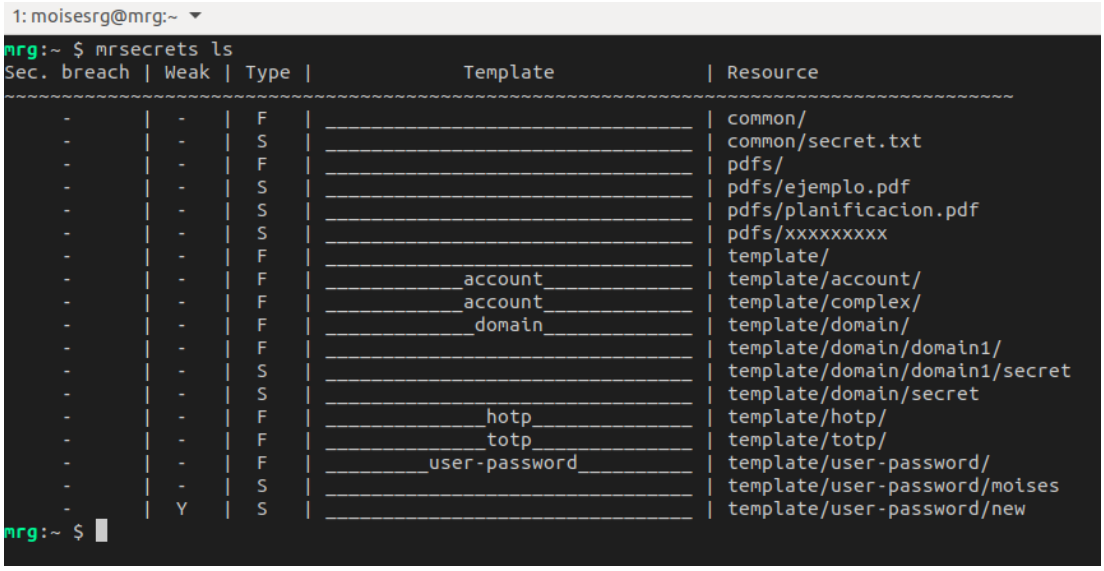

Figura 5.3: Listado de recursos

  - ***Sec. breach*:** saber si un secreto contiene un dominio que ha sufrido alguna brecha de seguridad desde la última edición del secreto.
  - ***Weak*:** saber si una clave es suficientemente robusta teniendo en cuenta las reglas que determinan si una clave es robusta o no.
  - ***Type*:** identificar si un recurso es una carpeta o un secreto.
  - ***Template*:** conocer la plantilla que tiene asociada una carpeta.
  - ***Resource*:** identificador del recurso.

Capítulo 6

Conclusiones

Tras haber analizado las herramientas actuales, tanto *open-source* como comerciales (capítulo 2), planteado un diseño de la aplicación e implementado la misma, he desarrollado la aplicación **MrSecrets**, un gestor de secretos polivalente, libre y que mejora aspectos como la capacidad de almacenar no solo contraseñas sino también archivos de propósito general, permite una gestión sencilla pero muy potente a la hora de asignar miembros a grupos y permisos tanto a nivel de usuario individual como a nivel de grupo ofreciendo un gran nivel de detalle, implementa de forma nativa soporte para generación de TOTPs y HOTPs, es compatible con sistemas Unix, etc.

Como se ha podido observar a lo largo de este documento, tanto los objetivos principales como los secundarios, propuestos en el capítulo 1, han sido cumplidos dando como resultado un nuevo gestor de secretos.

La nueva herramienta es plenamente funcional y está siendo puesta a prueba por el equipo de Allenta de cara a una migración a la misma lo antes posible. Esta aplicación permite a los usuarios que requieran de un gestor especializado en el almacenamiento de todo tipo de secretos y que ofrezca flexibilidad a la hora de gestionar grupos y asignar permisos. Además, tras haber superado un período de prueba prudencial que permita asegurar un funcionamiento correcto, el proyecto será publicado en GitHub bajo la licencia libre GPL. El objetivo de hacer público el proyecto bajo una licencia libre, es ocupar un hueco en los gestores *open-source* que actualmente se encuentra vacío, que se corresponde con un gestor *open-source* basado en GPG y cuya funcionalidad este pensada para cubrir las necesidades de compartición de secretos entre grupos de trabajo.

La herramienta ha sido validada en colaboración con el equipo de Allenta, utilizando diversos sistemas Linux (Ubuntu y Debian) y macOS mediante el uso de la misma por distintos usuarios. Estos la han utilizado para crear, editar, borrar y acceder a los secretos exitentes, y han almacenado archivos de configuración, claves de servicios, scripts de configuración, certificados, etc.

Personalmente este proyecto me ha aportado una gran cantidad de conocimientos relacionados con diversas disciplinas como son: la seguridad, la programación y el diseño de arquitecturas así como su administración. Además, me ha permitido reforzar las competencias aprendidas en cuanto a la programación en Python mediante el uso del *framework* Django con el que ya había trabajado en ciertas ocasiones a lo largo del grado y a realizar integraciones con otros servicios ya existentes mediante por ejemplo el acceso a APIs de terceros.

Por otra parte, como ya se ha mencionado, el proyecto no se ha basado únicamente en el desarrollo de la aplicación, sino que además, se ha tenido que diseñar, desplegar y configurar una infraestructura específica para ofrecer soporte al gestor de secretos. Gracias a esta parte he descubierto muchas tecnologías interesantes y he obtenido un pensamiento más crítico y analítico de cara a la elección de infraestructura a la hora de diseñar un entorno tanto de desarrollo como de producción.

Por último, en cuanto al estándar PGP (*Pretty Good Privacy*) y las librerías existentes para su integración, el desarrollo de este proyecto me ha permitido corroborar que a día de hoy es un sistema criptográfico anticuado, innecesariamente complejo, y que sería recomendable abandonar en favor de implementaciones criptográficas mucho más actuales como libsodium [38]. Su uso vino dado por los requisitos establecidos al inicio y debido a que en la empresa de mi director Carlos, se utiliza PGP internamente, tanto en su antiguo gestor de contraseñas, como en otras herramientas internas, si bien mi recomendación es la evolución hacia estándares más actuales.

Capítulo 7

Trabajos futuros

Tras terminar el proyecto se ha realizado un repaso completo del mismo y se han identificado una serie de mejoras que ayudarían a aumentar la riqueza y la usabilidad del gestor de secretos. Algunos de estos puntos se han tenido en cuenta a la hora de realizar el proyecto pero no han llegado a cumplirse debido a la dificultad de los mismos teniendo en cuenta el tiempo disponible. Es decir, se ha preferido priorizar el correcto funcionamiento y el cumplimiento de los objetivos establecidos sobre el añadido de mayor eficiencia o funcionalidad.

- Con respecto al **elevado número de consultas a base de datos** ya mencionado en el capítulo 5, lo ideal sería disponer de una capa de caché, integrada con el ORM de Django, y apoyada en Redis aprovechando la infraestructura ya existente. Implementar de forma manual la lógica para la capa de caché es una tarea muy compleja y actualmente existen aplicaciones que solucionan esta situación. Un ejemplo sería **Cacheops** [39], esta solución utiliza Redis como *backend* para el ORM de Django automatizando el cacheo de consultas y la invalidación automática de datos previamente cacheados.

- La **paginación de los recursos** es otra característica que puede ser mejorada. Actualmente, debido a la complejidad de la jerarquía, el cálculo de permisos para el usuario que realiza la petición obliga a tener la jerarquía de recursos construida para poder recuperar solo aquellos a los que tiene acceso. Por este motivo, el uso de una capa caché que permita almacenarla de forma eficiente en memoria, evitando recuperarlos de base de datos en cada petición paginada, aumentaría la eficiencia y velocidad de respuesta del gestor.

- **El concepto de plantilla** actual incluye el uso de **JSON Schema** para la definición de secretos que comparten una misma estructura de datos. Actualmente las plantillas están limitadas debido a que simplemente se utilizan los tipos básicos definidos por el propio JSON Schema. La mejora consistiría en definir nuevos tipos de datos como por ejemplo el tipo TOTP o HOTP que actualmente se identifican mediante la búsqueda de

un campo en las plantillas. Esta mejora permite un nivel de personalización mucho más amplio y útil de cara al usuario final.

- Sería interesante utilizar alguna **herramienta de tipado de Python como Mypy [40]**. Debido a la naturaleza de Python y al uso que este hace del tipado dinámico, esta herramienta permite hacer un tipado estático previo a la fase de ejecución. Para esto se añadirán anotaciones al código ayudando a reducir los errores de implementación, a cambio de realizar una cantidad de trabajo previo considerable. Es un añadido que aporta mucho al proyecto pero que, debido al esfuerzo y tiempo que supone además de que aún no es un proyecto muy utilizado y probado por la comunidad, se ha decidido no utilizar todavía.
- Tendría interés explorar la **integración con alguna de las herramientas mencionadas en el capítulo 5 sección 5.2 para mejorar la protección de la clave simétrica** empleada para acceder a la clave privada del servicio.
- Por último, una modificación que aportaría mucho valor al gestor sería el **reemplazo de GPG** por un sistema criptográfico más moderno. Una solución muy interesante sería el uso de la librería **libsodium**, que permite cifrar, descifrar, firmar, validar firmas, etc.

Apéndices

Apéndice A

Glosario de acrónimos

ACL *Access Control List.*

API *Application Programming Interface.*

AWS *Amazon Web Services.*

BD *Base de Datos.*

CLI *Command Line Interface.*

EER *Enhanced Entity-Relationship.*

GPG *GNU Privacy Guard.*

HOTP *HMAC-based One-Time Password.*

NFS *Network File System.*

ORM *Object-Relational Mapping.*

OTP *One-Time Password.*

PGP *Pretty Good Privacy.*

REST *Representational State Transfer.*

SDK *Software Development Kit.*

SPOF *Single Point Of Failure.*

SSH *Secure SHell.*

SSO *Single Sign-On.*

TLS *Transport Layer Security.*

TOTP *Time-based One-Time Password.*

VPN *Virtual Private Network.*

WLAN *Wireless Local Area Network.*

Apéndice B

Glosario de términos

Broker Programa que actúa como intermediario entre dos sistemas realizando la traducción del protocolo de mensaje del emisor al protocolo de mensaje del receptor.

Claves SSH Conjunto de claves compuesto por una clave pública y una clave privada que implican el uso de criptografía asimétrica para cifrar, descifrar, etc. Permiten realizar conexiones SSH mediante el almacenamiento de la clave pública en el servidor al que se quiere acceder y la clave privada en el equipo desde el que se realizará la conexión.

Criptografía asimétrica Sistema criptográfico que basa su seguridad en el uso de dos claves, una clave privada que solo conoce el propietario y otra pública que puede ser accesible por cualquier usuario. Es importante destacar que es imposible generar dos pares de claves iguales.

Criptografía simétrica Sistema criptográfico que se basa en la utilización de una clave secreta que se utiliza tanto para cifrar como para descifrar. Para acceder al contenido cifrado es necesario conocer la clave secreta por lo que si se quiere permitir a otro usuario acceder a los datos cifrados, este debe conocer la clave.

Timestamp Marca de tiempo que se utiliza en informática para conocer el momento en el que se genera, cifra, firma, etc. un cierto mensaje. Muy utilizado en todos los ambientes de la informática. En sistemas UNIX, es un número que se corresponde con la cantidad de segundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC.

Bibliografía

[1] "Reposirotio GitHub de Gopass." [Online]. Available: https://github.com/gopasspw/gopass

[2] "KeePass web." [Online]. Available: https://keepass.info/index.html

[3] "1Password web." [Online]. Available: https://1password.com/

[4] "LastPass web." [Online]. Available: https://www.lastpass.com/es

[5] "Dashlane web." [Online]. Available: https://www.dashlane.com/

[6] "Allenta Consulting S.L. web." [Online]. Available: https://allenta.com/es

[7] "Python Software Fundation." [Online]. Available: https://www.python.org/

[8] "The web framework Django." [Online]. Available: https://www.djangoproject.com/

[9] H. Finney, L. Donnerhacke, J. Callas, R. L. Thayer, and D. Shaw, "OpenPGP Message Format," RFC 4880, Tech. Rep. 4880, Nov. 2007. [Online]. Available: https://tools.ietf.org/html/rfc4880

[10] Paolo Gasti and Kasper B. Rasmussen, University of California, Irvine, "On The Security of Password Manager Database Formats." [Online]. Available: https://www.cs.ox.ac.uk/files/6487/pwvault.pdf

[11] "Password-Store web." [Online]. Available: https://www.passwordstore.org/

[12] "Explicación de los formatos de fichero kdb y kdbx en KeePass." [Online]. Available: https://gist.github.com/lgg/e6ccc6e212d18dd2ecd8a8c116fb1e45

[13] "Bitwarden web." [Online]. Available: https://bitwarden.com/

[14] "Bitwarden Completes Third-party Security Audit." [Online]. Available: https://blog.bitwarden.com/bitwarden-completes-third-party-security-audit-c1cc81b6d33

[15] "Bitwarden - HackerOne." [Online]. Available: https://hackerone.com/bitwarden

[16] "Bitwarden - Using file attachments." [Online]. Available: https://help.bitwarden.com/article/attachments/

[17] "LessPass web." [Online]. Available: https://lesspass.com/#/

[18] "GitLab security practices, use of 1Password." [Online]. Available: https://about.gitlab.com/handbook/security/

[19] "1Password, synchronization options security." [Online]. Available: https://support.1password.com/sync-options-security/

[20] "LastPass is scrambling to fix another serious vulnerability." [Online]. Available: https://www.pcworld.com/article/3185731/lastpass-is-scrambling-to-fix-another-serious-vulnerability.html

[21] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song, University of California, Berkeley, "The Emperor's New Password Manager: Security Analysis of Web-based Password Managers." [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-li-zhiwei.pdf

[22] D. Silver, S. Jana, D. Boneh, E. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 449–464. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver

[23] "Redis web." [Online]. Available: https://redis.io/

[24] "Celery web." [Online]. Available: http://www.celeryproject.org/

[25] "Sentry web." [Online]. Available: https://sentry.io/welcome/

[26] "Celery web." [Online]. Available: https://docs.celeryproject.org/en/latest/userguide/periodic-tasks.html

[27] "Documentación del uso de las réplicas en Redis." [Online]. Available: https://redis.io/topics/replication

[28] "Documentación de Redis Sentinel." [Online]. Available: https://redis.io/topics/sentinel

[29] "High-Availability MySQL cluster." [Online]. Available: https://dev.mysql.com/doc/mysql-ha-scalability/en/ha-overview.html

[30] "GlusterFS." [Online]. Available: https://www.gluster.org/

[31] "Bitbucket." [Online]. Available: https://bitbucket.org/

[32] "Python-gnupg library documentation." [Online]. Available: https://gnupg.readthedocs.io/en/latest/

[33] "Oracle VM VirtualBox." [Online]. Available: https://www.virtualbox.org/

[34] "Vagrant." [Online]. Available: https://www.vagrantup.com/

[35] "Ansible." [Online]. Available: https://www.ansible.com/

[36] "AWS Key Management Service." [Online]. Available: https://aws.amazon.com/es/kms/

[37] "Oracle Database Vault." [Online]. Available: https://www.oracle.com/technetwork/es/articles/idm/proteger-datos-oracle-db-vault-2492703-esa.html

[38] "Libsodium documentation." [Online]. Available: https://download.libsodium.org/doc/

[39] "Cacheops." [Online]. Available: https://github.com/Suor/django-cacheops

[40] "Mypy." [Online]. Available: http://mypy-lang.org/